USER MANUAL

DMC-30000

Manual Rev. 1.0e

By Galil Motion Control, Inc.

Galil Motion Control, Inc.
270 Technology Way
Rocklin, California 95765
Phone: (916) 626-0101
Fax: (916) 626-0102
E-mail Address: support@galilmc.com
URL: www.galilmc.com

Date: 08/12

Using This Manual

This user manual provides information for proper operation of the DMC-30000 controller. A separate supplemental manual, the Command Reference, contains a description of the commands available for use with this controller. It is recommended that the user download the latest version of the Command Reference and User Manual from the Galil Website.

http://www.galilmc.com/support/manuals.php

Your DMC-30000 motion controller has been designed to work with both servo and stepper type motors. Installation and system setup will vary depending upon whether the controller will be used with stepper motors or servo motors. To make finding the appropriate instructions faster and easier, icons will be next to any information that applies exclusively to one type of system. Otherwise, assume that the instructions apply to all types of systems. The icon legend is shown below.

Attention: Pertains to servo motor use.

Attention: Pertains to stepper motor use.

WARNING: Machinery in motion can be dangerous! It is the responsibility of the user to design effective error handling and safety protection as part of the machinery. Galil shall not be liable or responsible for any incidental or consequential damages.

Contents

Chapter 1 Overview

Introduction

The DMC-30000 Series is Galil's latest generation single-axis motion controller. It uses a 32-bit RISC processor to provide higher speed than older models. The DMC-30000 is available as a compact card-level or box-level unit and connects to a stepper or servo motor amplifier of any power range. Or, the DMC-300xx can be purchased with an internal drives which minimize space, cost and wiring. The motion controller operates stand-alone or can be networked to a PC via Ethernet.

Features include PID compensation with both velocity and acceleration feed-forward, program memory with multitasking for concurrent execution of multiple programs, and uncommitted optically isolated inputs and outputs for synchronizing motion with external events. Modes of motion include point-to-point positioning, jogging, contouring, PVT, electronic gearing and electronic cam. Like all Galil motion controllers, these controllers use a simple, English-like command language which makes them very easy to program. GalilTools software further simplifies system set-up with "one button" servo tuning and real-time display of position and velocity information.

Designed to solve complex motion problems, the DMC-30000 can be used for applications involving jogging, point-to-point positioning, vector positioning, electronic gearing, multiple move sequences, contouring and a PVT Mode. The controller eliminates jerk by programmable acceleration and deceleration with profile smoothing. For smooth following of complex contours, the DMC-30000 provides continuous vector feed of an infinite number of linear and arc segments. The controller also features electronic gearing with multiple master axes as well as gantry mode operation.

For synchronization with outside events, the DMC-30000 provides uncommitted I/O, including 8 optoisolated digital inputs, 4 optically isolated outputs, 2 analog inputs for interface to joysticks, sensors, and pressure transducers and 1 uncommitted analog output. Further I/O is available if the auxiliary encoders are not being used (2 inputs / each axis). Dedicated optoisolated inputs are provided for forward and reverse limits, abort, home, and definable input interrupts.

Commands are sent in ASCII. Additional software is available for automatic-tuning, trajectory viewing on a PC screen, and program development using many environments such as Visual Basic, C, C++ etc. Drivers for Windows XP, Vista and 7 (32 & 64 bit) as well as Linux are available.

Overview of Part Numbers

The below table shows the main categories of the DMC-30000 controller family. Other options and modifications are available, see Ordering Options for the DMC-30000 and the DMC-30000 part number generator (http://www.galilmc.com/products/dmc-300xx-part-number.php) for more information.

Controller Model	Description
DMC-30010-Card	Single Axis Controller card (Requires 5V,+/-12VDC Triple Supply)
DMC-30011-Card	Single Axis Controller card with DC-to-DC
DMC-30010-Box	Single Axis Controller box (Requires 5V,+/-12VDC Triple Supply)
DMC-30011-Box	Single Axis Controller box with DC-to-DC
DMC-30012-Box	Single Axis Controller box with internal 800W sine drive with 20-80VDC Input See A1 – DMC-30012 for Amplifier Specifications.
DMC-30016-Box	Single Axis Controller box with internal 1.4A stepper driver. See A2 – DMC-30016 for Driver Specifications.
DMC-30017-Box	Single Axis Controller box with internal 6Amp/phase 256th microstepping stepper driver or internal 800W side drive with 20-80VDC Input. See A3 – DMC-30017 for Driver Specifications.

Overview of Motor Types

The DMC-30000 can provide the following types of motor control:

1. Standard servo motors with +/- 10 volt command signals
2. Step motors with step and direction signals
3. Other actuators such as hydraulics and ceramic motors - For more information, contact Galil.

The user can configure each axis for any combination of motor types, providing maximum flexibility.

Standard Servo Motor with +/- 10 Volt Command Signal

The DMC-30000 achieves superior precision through use of a 16-Bit motor command output DAC and a sophisticated PID filter that features velocity and acceleration feed-forward, an extra pole filter and integration limits.

The controller is configured by the factory for standard servo motor operation. In this configuration, the controller provides an analog signal (+/-10 volts) to connect to a servo amplifier. This connection is described in Chapter 2.

Stepper Motor with Step and Direction Signals

The DMC-30000 can control stepper motors. In this mode, the controller provides two signals to connect to the stepper motor: Step and Direction. For stepper motor operation, the controller does not require an encoder and operates the stepper motor in an open loop fashion. Chapter 2 describes the proper connection and procedure for using stepper motors.

If encoders are available on the stepper motor, Galil's Stepper Position Maintenance Mode may be used for automatic monitoring and correction of the stepper position. See Stepper Position Maintenance Mode (SPM) in Chapter 6 for more information.

Overview of External Amplifiers

The amplifiers should be suitable for the motor and may be linear or pulse-width-modulated. An amplifier may have current feedback, voltage feedback or velocity feedback.

Amplifiers in Current Mode

Amplifiers in current mode should accept an analog command signal in the +/-10 volt range. The amplifier gain should be set such that a +10V command will generate the maximum required current. For example, if the motor peak current is 10A, the amplifier gain should be 1 A/V.

Amplifiers in Velocity Mode

For velocity mode amplifiers, a command signal of 10 volts should run the motor at the maximum required speed. The velocity gain should be set such that an input signal of 10V runs the motor at the maximum required speed.

Stepper Motor Amplifiers

For step motors, the amplifiers should accept step and direction signals.

Overview of the Galil Amplifiers

With the DMC-30000 Galil offers amplifiers that are integrated into the same enclosure as the controller. Using the Galil Amplifier provides a simple straightforward motion control solution in one box.

DMC-30012 (DMC-30000 with 800W Sinusoidal Amplifier)

The DMC-30012 (A1 – DMC-30012) provides an amplifier that drives motors operating at 20–80 VDC, up to 10 Amps continuous, 15 Amps peak. The gain settings of the amplifier are user-programmable at 0.4 Amp/Volt, 0.8 Amp/Volt and 1.6 Amp/Volt. The switching frequency is 33 kHz. The amplifier offers protection for over-voltage, under-voltage, over-current, and short-circuit. The SR90 – SR-49000 Shunt Regulator Option is also available for the DMC-30012.

DMC-30016 (DMC-30000 with 1.4 Amp stepper driver)

The DMC-30016 (A2 – DMC-30016) includes a microstepping drive for operating two-phase bipolar stepper motors.

The DMC-30016 drive operates a two-phase bipolar stepper motor in full-step, half-step, 1/4 step or 1/16 step. It is user configurable from 0.5A to 1.4A per phase in ~7 mA increments at 12-30VDC. The dimensions of the DMC-30016 controller and drive package are 3.9" x 5.0" x 1.5", and no external heatsink is required.

DMC-30017 (DMC-30000 with 6Amp stepper driver or 800W Sinusoidal Amplifier)

The DMC-30017 (A3 – DMC-30017) includes a microstepping drive for operating two-phase bipolar stepper motors, the drive can also be configured for a sinusoidally commutated, PWM amplifier for driving brushed or brushless servo motors.

Micro-stepping Drive: The micro-stepping drive produces 256 microsteps per full step or 1024 steps per full cycle which results in 51,200 steps/rev for a standard 200-step motor. The maximum step rate generated by the controller is 3,000,000 microsteps/second. The DMC-30017 can drive stepper motors at up to 6 Amps at 20-80VDC. There are four selectable current gains: 0.75 A, 1.5 A, 3 A and 6A. A selectable low current mode reduces the current by 75% when the motor is not in motion.

Sinusoidally Commutated Amplifier: When set to servo mode, the DMC-30017 has the same specs as the DMC-30012.

DMC-30000 Functional Elements

The DMC-30000 circuitry can be divided into the following functional groups as shown in Figure 1.1 and discussed below.

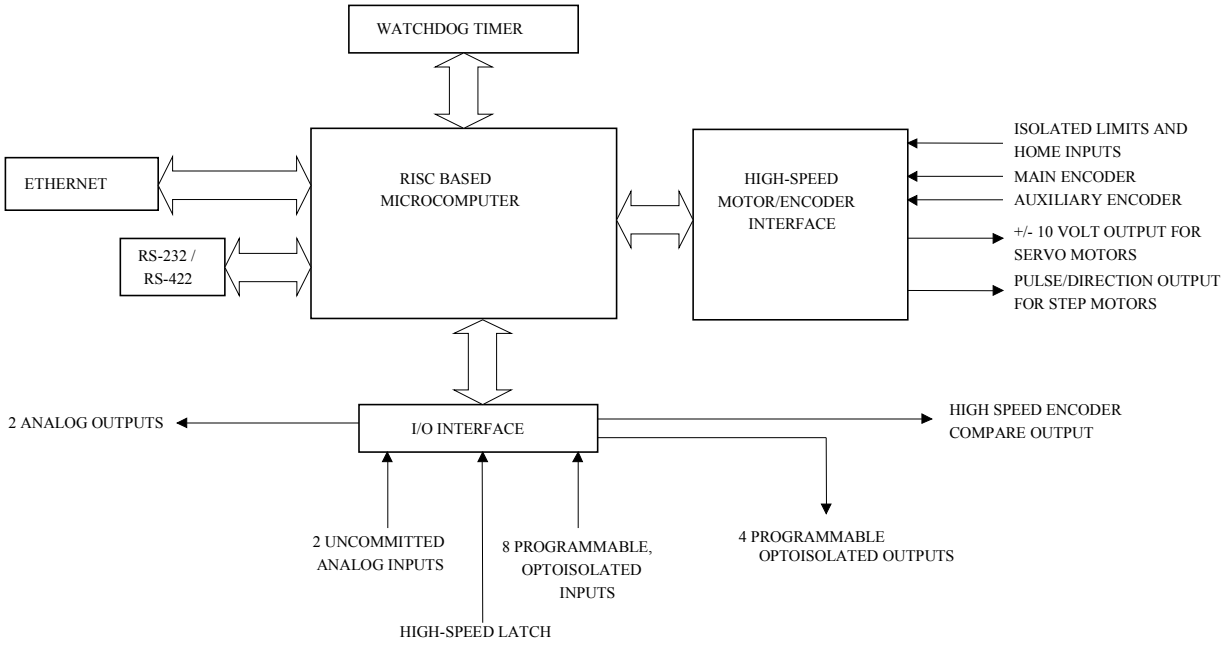


Figure 1.1: DMC-30000 Functional Elements

Microcomputer Section

The main processing unit of the controller is a specialized Microcomputer with RAM and Flash FLASH. The RAM provides memory for variables, array elements, and application programs. The flash FLASH provides non-volatile storage of variables, programs, and arrays. The Flash also contains the firmware of the controller, which is field upgradeable.

Motor Interface

Galil's GL-1800 custom, sub-micron gate array performs quadrature decoding of each encoder at up to 15 MHz. For standard servo operation, the controller generates a +/-10 volt analog signal (16 Bit DAC). For stepper motor operation, the controller generates a step and direction signal.

Communication

The communication interface with the DMC-30000 consists of a daisy-chainable Ethernet 100 Base-T port and a 115kbaud RS-232 programming port.

General I/O

The DMC-30000 provides interface circuitry for 8 bi-directional, optoisolated inputs, 4 optoisolated outputs and 2 analog inputs with 12-Bit ADC (16-Bit optional). Unused auxiliary encoder inputs may also be used as additional

inputs (2 inputs). The general inputs as well as the index pulse can also be used as high speed latches for each axis. A high speed encoder compare output is also provided.

System Elements

As shown in Figure 1.2, the DMC-30000 is part of a motion control system which includes amplifiers, motors and encoders. These elements are described below.

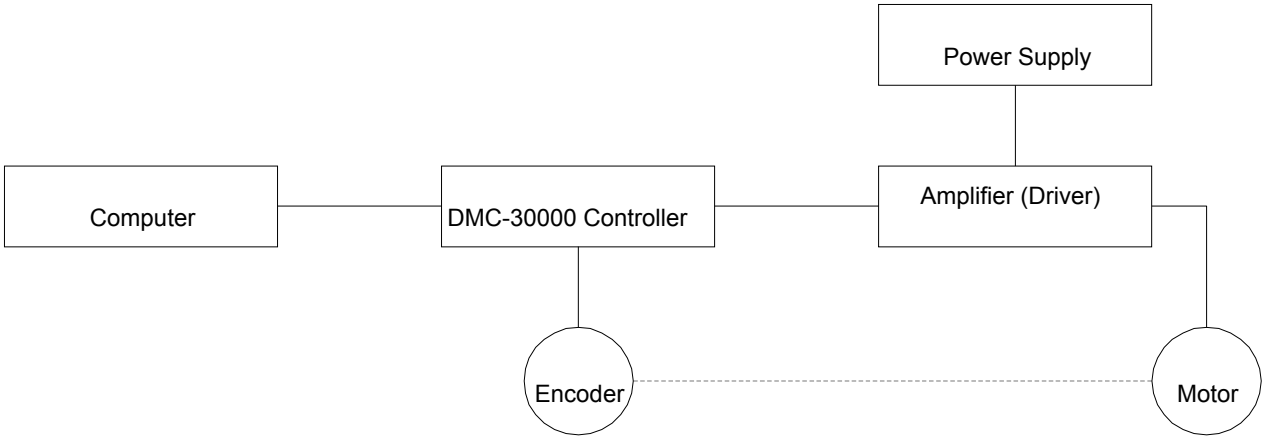


Figure 1.2: Elements of Servo Systems

Motor

A motor converts current into torque which produces motion. Each axis of motion requires a motor sized properly to move the load at the required speed and acceleration. (Galil's MotorSizer Web tool can help you with motor sizing: www.galilmc.com/support/motorsizer)

The motor may be a step or servo motor and can be brush-type or brushless, rotary or linear. For step motors, the controller can be configured to control full-step, half-step, or microstep drives. An encoder is not required when step motors are used.

Other motors and devices such as Ultrasonic Ceramic motors and voice coils can be controlled with the DMC-30000.

Amplifier (Driver)

The power amplifier converts a +/-10 volt signal from the controller into current to drive the motor. For stepper motors, the amplifier converts step and direction signals into current. The amplifier should be sized properly to meet the power requirements of the motor. For brushless motors, an amplifier that provides electronic commutation is required or the controller must be configured to provide sinusoidal commutation. The amplifiers may be either pulse-width-modulated (PWM) or linear. They may also be configured for operation with or without a tachometer. For current amplifiers, the amplifier gain should be set such that a 10 volt command generates the maximum required current. For example, if the motor peak current is 10A, the amplifier gain should be 1 A/V. For velocity mode amplifiers, 10 volts should run the motor at the maximum speed.

Galil offers amplifiers that is integrated into the same enclosure as the DMC-30000. See the A1 – DMC-30012, A2 – DMC-30016 and the A3 – DMC-30017 sections in the Appendices or http://galilmc.com/products/dmc-300xx.php for more information.

Encoder

An encoder translates motion into electrical pulses which are fed back into the controller. The DMC-30000 accepts feedback from either a rotary or linear encoder. Typical encoders provide two channels in quadrature, known as MA and MB. This type of encoder is known as a quadrature encoder. Quadrature encoders may be either single-ended (MA and MB) or differential (MA+, MA- and MB+, MB-). The DMC-30000 decodes either type into quadrature states or four times the number of cycles. Encoders may also have a third channel (or index) for synchronization.

The DMC-30000 can be ordered with 120Ω termination resistors installed on the encoder inputs. See the Ordering Options for the in the Appendix for more information.

The DMC-30000 can also interface to encoders with pulse and direction signals. Refer to the "CE" command in the command reference for details.

There is no limit on encoder line density; however, the input frequency to the controller must not exceed 3,750,000 full encoder cycles/second (15,000,000 quadrature counts/sec). For example, if the encoder line density is 10,000 cycles per inch, the maximum speed is 200 inches/second. If higher encoder frequency is required, please consult the factory.

The standard encoder voltage level is TTL (0-5v), however, voltage levels up to 12 Volts are acceptable. (If using differential signals, 12 Volts can be input directly to the DMC-30000. Single-ended 12 Volt signals require a bias voltage input to the complementary inputs).

The DMC-30000 can accept analog feedback (+/-10v) instead of an encoder for any axis. For more information see the command AF in the command reference.

To interface with other types of position sensors such as absolute encoders, Galil can customize the controller and command set. Please contact Galil to talk to one of our applications engineers about your particular system requirements.

Watch Dog Timer

The DMC-30000 provides an internal watch dog timer which checks for proper microprocessor operation. The timer toggles the Amplifier Enable Output (AEN) which can be used to switch the amplifiers off in the event of a serious DMC-30000 failure. The AEN output is normally high. During power-up and if the microprocessor ceases to function properly, the AEN output will go low. The error light will also turn on at this stage. A reset is required to restore the DMC-30000 to normal operation. Consult the factory for a Return Materials Authorization (RMA) Number if your DMC-30000 is damaged.

Chapter 2 Getting Started

DMC-30010-CARD Dimensions

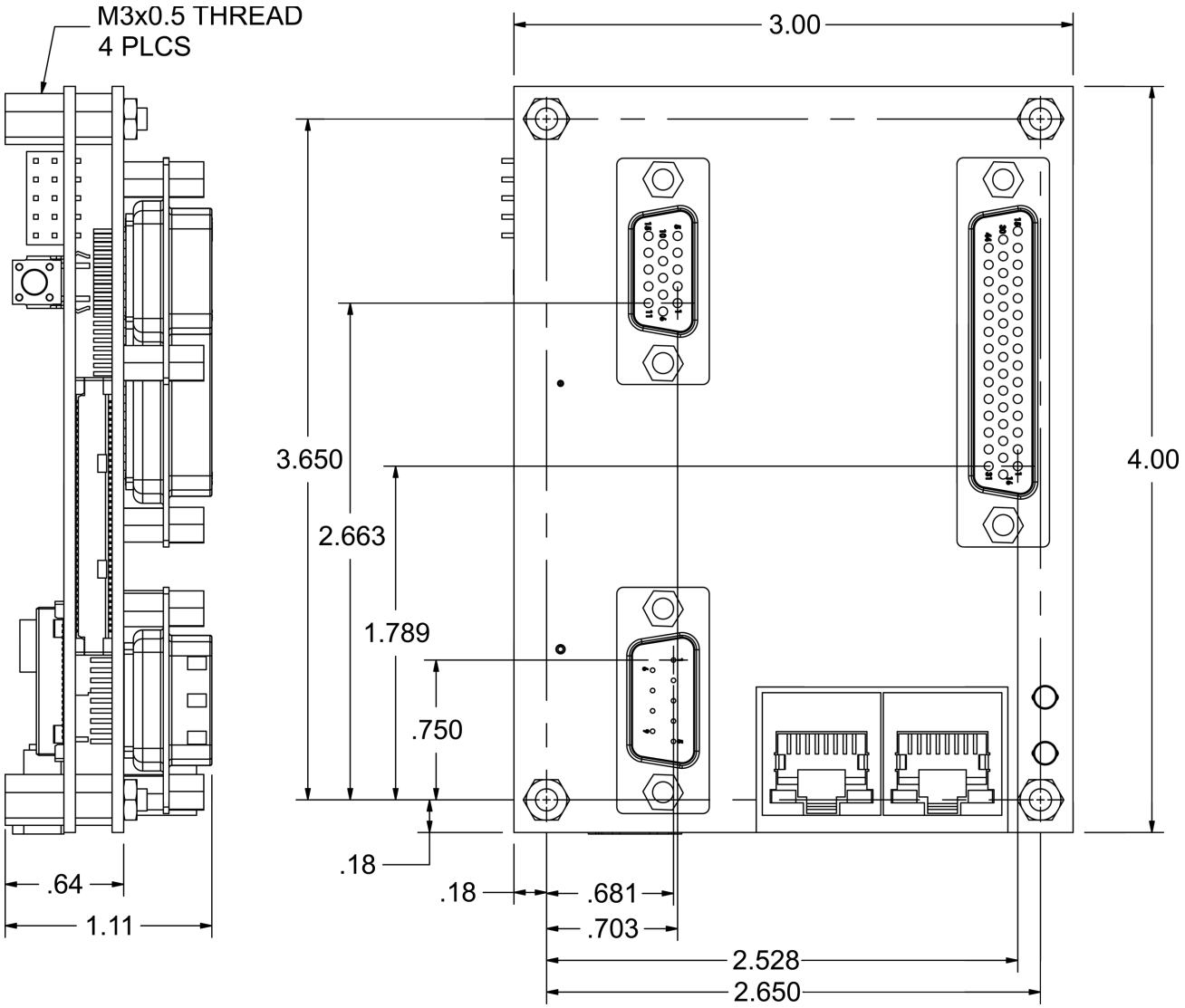


Figure 2.1: DMC-30010-CARD Dimensions

DMC-30011-CARD Dimensions

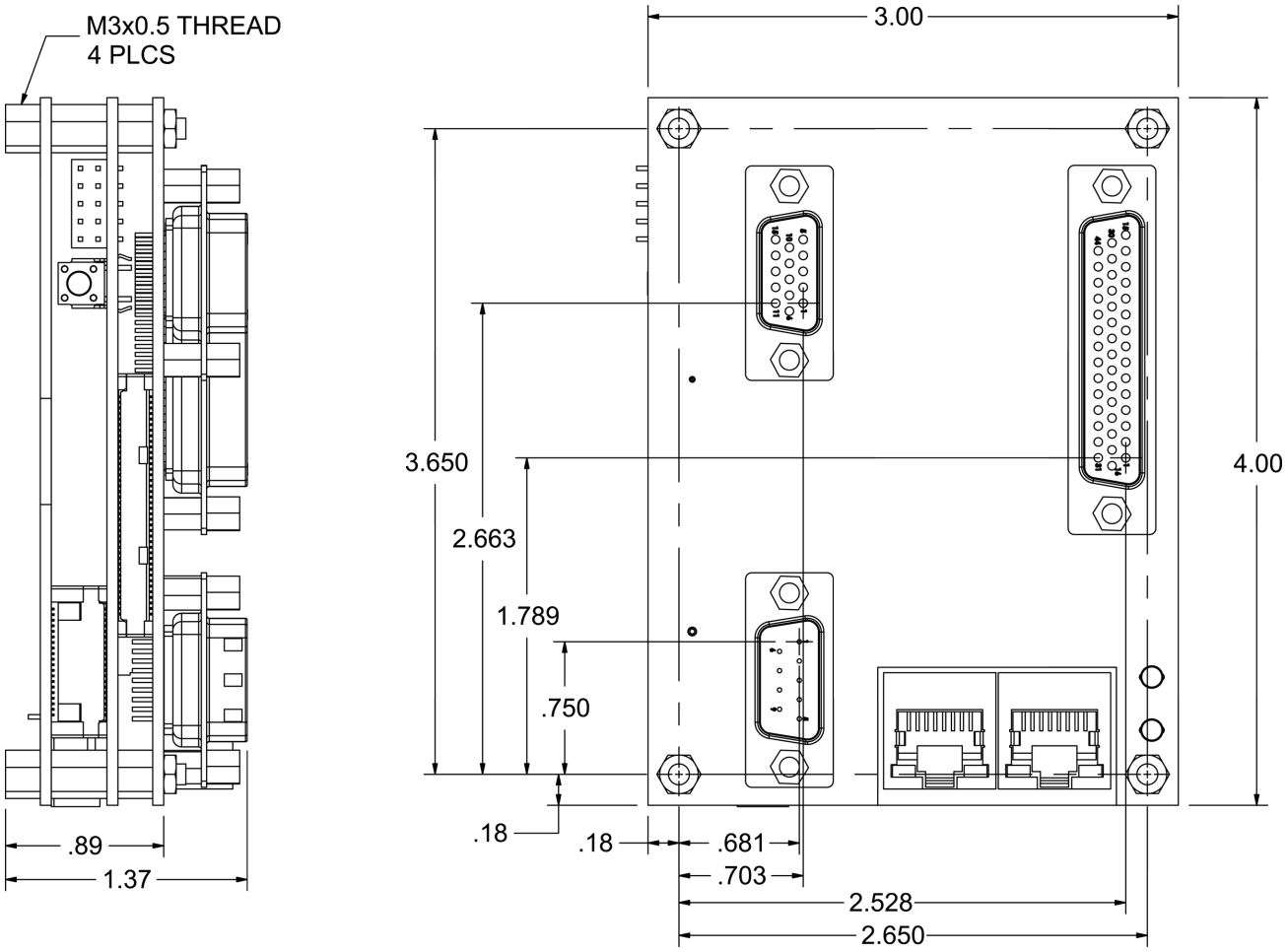


Figure 2.2: DMC-30011-CARD Dimensions

DMC-30010-BOX and DMC-30011-BOX Dimensions

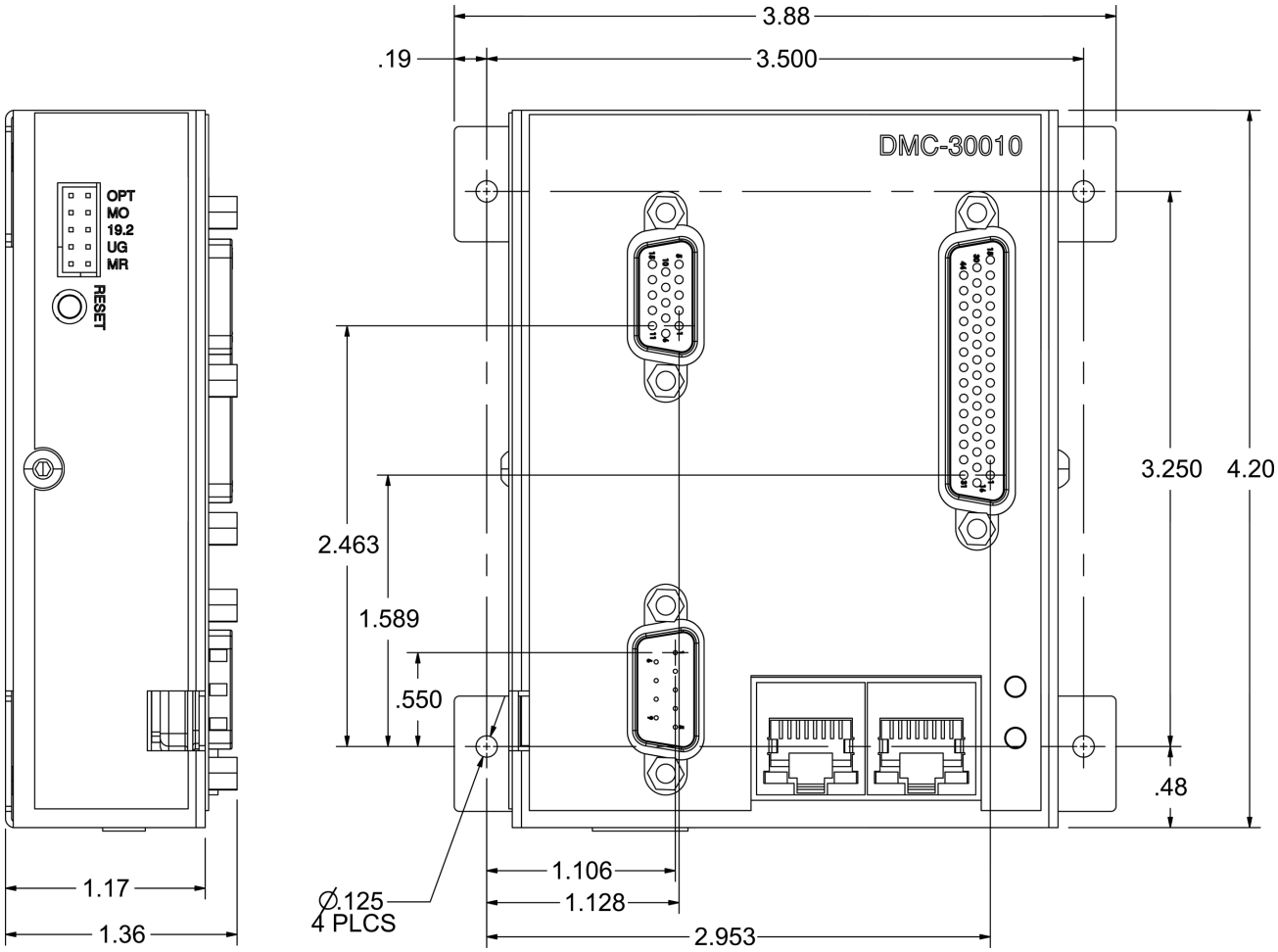


Figure 2.3: DMC-30010-BOX and DMC-30011-BOX Dimensions

DMC-30012-BOX, DMC-30016-BOX and DMC-30017-BOX Dimensions

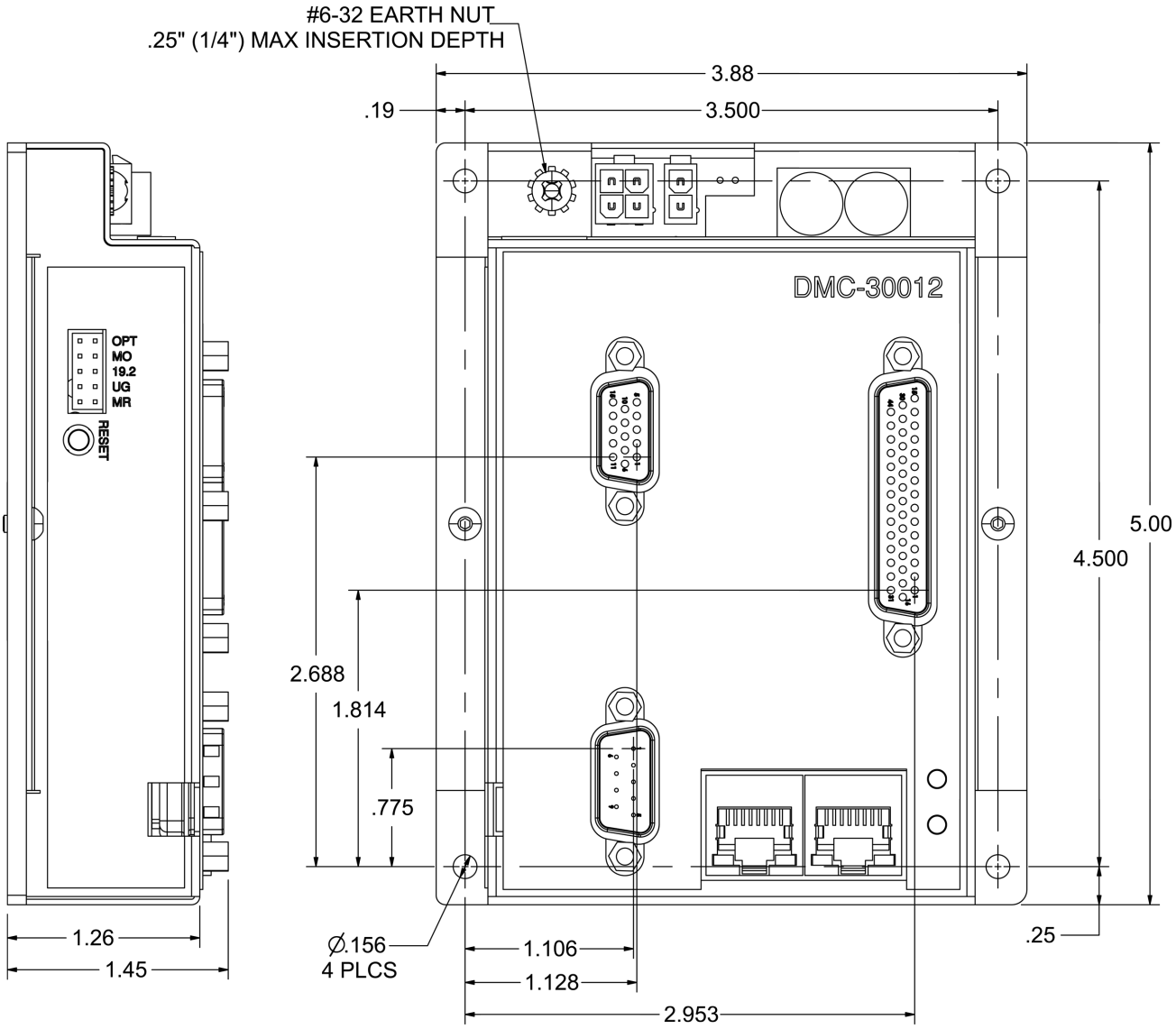


Figure 2.4: DMC-30012-BOX, DMC-30016-BOX and DMC-30017-BOX Dimensions

DMC-30000 Mounting Instructions

-CARD

All 4 standoff locations must be used when mounting the -CARD version of the DMC-30000 controllers. See Figure 2.1 and Figure 2.2 for mounting screw sizing and locations.

-BOX

All 4 mounting holes should be used to mount the controller to a secure base. See Figure 2.3 and Figure 2.4 for mounting hole locations and sizes.

DMC-30012, DMC-30016 and DMC-30017

The bases for the DMC-30012, DMC-30016 and DMC-30017 are used as the heat-sync for the internal amplifier. The DMC-30012 and DMC-30017 must be mounted to an external heat-sync for high duty cycle applications.

Elements You Need

For a complete system, Galil recommends the following elements:

1. DMC-30000
2. Motor Amplifier (Integrated when using DMC-30012, DMC-30016, DMC-30017)
3. Power Supply for Amplifiers and Controller
4. Brush or Brushless Servo motors with Optical Encoders or stepper motors.
    a. Cables for connecting to the DMC-30000.
5. PC (Personal Computer - Serial or Ethernet for DMC-30000)
6. GalilTools, or GalilTools-Lite Software package

GalilTools is highly recommended for first time users of the DMC-30000.

Installing the DMC-30000

Installation of a complete, operational DMC-30000 system consists of 8 steps.

Step 1. Determine overall motor configuration.

Step 2. Install Jumpers on the DMC-30000.

Step 3. Install the communications software.

Step 4. Connect DC power to controller.

Step 5. Establish communications with the Galil Communication Software.

Step 6. Make connections to amplifier and encoder.

Step 7a. Connect standard servo motors.

Step 7b. Connect step motors.

Step 8. Tune the servo system

Step 1. Determine Overall Motor Configuration

Before setting up the motion control system, the user must determine the desired motor configuration. The DMC-30000 can control either a standard servo motor or stepper motor. Other types of actuators, such as hydraulics can also be controlled, please consult Galil.

The following configuration information is necessary to determine the proper motor configuration:

Standard Servo Motor Operation:

Unless ordered with stepper motor drivers or in a non-standard configuration, the DMC-30000 has been setup by the factory for standard servo motor operation providing an analog command signal of +/- 10V. No hardware or software configuration is required for standard servo motor operation.

Stepper Motor Operation

To configure the DMC-30000 for stepper motor operation, the controller requires that the command, MT, must be given. Further instruction for stepper motor connections are discussed in Step 7b.

Step 2. Install Jumpers on the DMC-30000

Motor Off Jumper

The state of the motor upon power up may be selected with the placement of a hardware jumper on the controller. With a jumper installed at the MO location, the controller will be powered up in the "motor off" state. The SH command will need to be issued in order for the motor to be enabled. With no jumper installed, the controller will immediately enable the motor upon power up. The MO command will need to be issued to turn the motor off, unless an error occurs that will turn the motors off. The MO jumper is located on JP1, the same block as the Master Reset (MR) and Upgrade (UG) jumpers.

Communications Jumpers for DMC-30000

The baud rate for RS-232 communication can be set with jumpers found on JP1 of the communication board (same set of jumpers where MO, MR and UG can be found). There are two options for baud rate, 19200, and 115200. To set the baud rate to the desired value, see Table 2.1 below.

19.2	BAUD RATE
ON	19200
OFF	115200

Table 2.1: Baud Rate settings for RS-232 communication

Master Reset and Upgrade Jumpers

JP1 on the main board contains two jumpers, MR and UG. The MR jumper is the Master Reset jumper. When MR is connected, the controller will perform a master reset upon PC power up or upon the reset input going low. Whenever the controller has a master reset, all programs, arrays, variables, and motion control parameters stored in FLASH will be ERASED.

The UG jumper enables the user to unconditionally update the controller's firmware. This jumper is not necessary for firmware updates when the controller is operating normally, but may be necessary in cases of corrupted FLASH. FLASH corruption should never occur, however, it is possible if there is a power fault during a firmware update. If FLASH corruption occurs, your controller may not operate properly. In this case, install the UG Jumper and use the update firmware function on the Galil Terminal to re-load the system firmware.

Step 3. Install the Communications Software

Using Windows XP, Vista and 7 (32 & 64 bit):

Install the Galil Software Products CD-ROM into your CD drive. A Galil .htm page should automatically appear with links to the software products. Select the correct version of GalilTools software for your particular operating system (32 or 64 bit) and click "Install…" Follow the installation procedure as outlined.

The most recent copy of the GalilTools software can be downloaded from the Galil website:

http://www.galilmc.com/products/software/galiltools.html

All other Galil software is also available for download at the Galil software downloads page:

http://www.galilmc.com/support/download.html

Using Linux (32 & 64 bit):

The GalilTools software package is fully compatible with a number of Linux distributions. See the GalilTools webpage and user manual for downloads and installation instructions.

http://www.galilmc.com/products/software/galiltools.html

Step 4. Connect Power to the Controller

See the Power Wiring Diagrams for the DMC-30000 section in the Appendices for information on wiring specific DMC-30000 configurations. Wiring diagrams are shown for:

DMC-30010-CARD,DMC-30010-BOX,DMC-30011-CARD,DMC-30011-BOX,DMC-30012-BOX, DMC-30016-BOX and DMC-30017-BOX and the DMC-30012-BOX(ISCNTL), DMC-30016-BOX(ISCNTL) and DMC-30017-BOX(ISCNTL).

A list off all of the motor and power connectors can be found in the Power Connectors for the DMC-30000 in the Appendices.

The DMC-30000 power should never be plugged in HOT. Always power down the power supply before installing or removing the power connector to the controller. If a disconnect or safety relay is required for an application, the disconnect switch or circuit breaker must be placed on the AC power (not DC input to the controller) and must be installed in a location that is in close proximity to the equipment, within easy reach of the operator, and must not block the installation of the controller. This is shown in Figure 2.5.

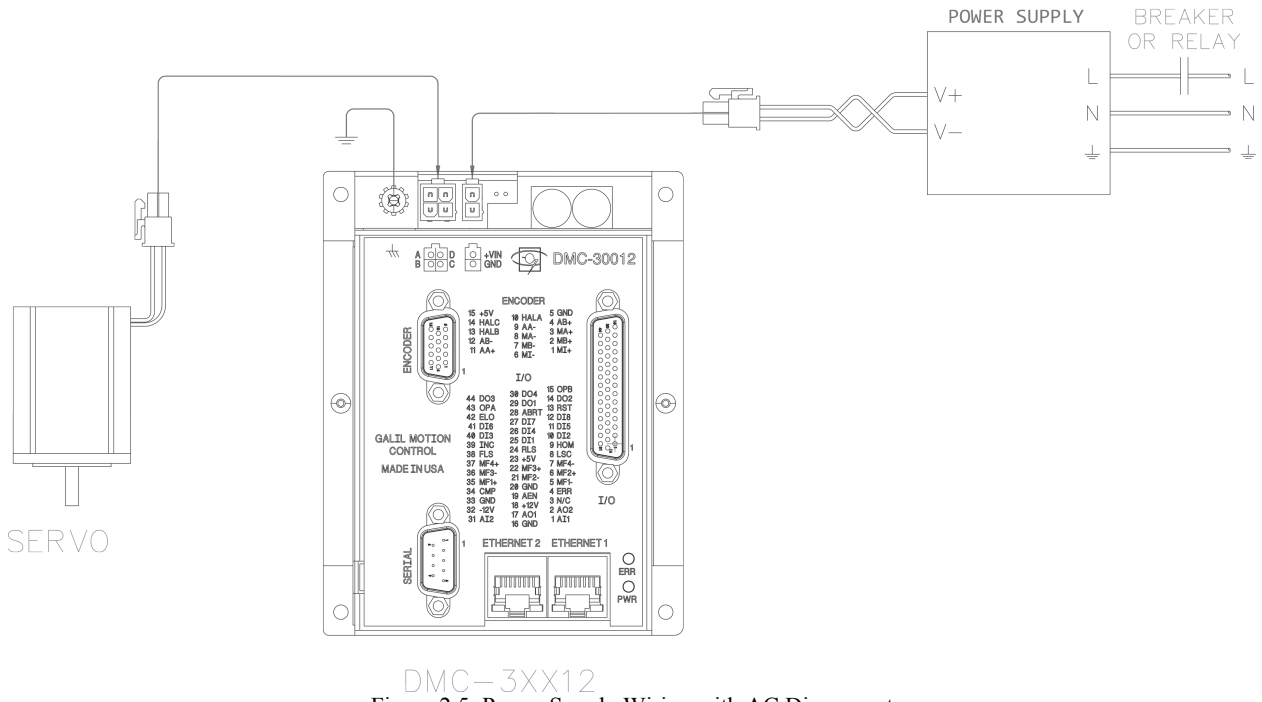


Figure 2.5: Power Supply Wiring with AC Disconnect

WARNING: Dangerous voltages, current, temperatures and energy levels exist in this product and the associated amplifiers and servo motor(s). Extreme caution should be exercised in the application of this equipment. Only qualified individuals should attempt to install, set up and operate this equipment. Never open the controller box when DC power is applied to it.

The green power (POWER) led indicator should go on when proper power is applied.

Step 5. Establish Communications with Galil Software

Communicating through an Ethernet connection

The DMC-30000 motion controller is equipped with DHCP. If the controller is connected to a DHCP enabled network, an IP address will automatically be assigned to the controller. See Ethernet Configuration in Chapter 4 for more information.

Using GalilTools Software for Windows

Registering controllers in the Windows registry is no longer required when using the GalilTools software package. A simple connection dialog box appears when the software is opened that shows all available controllers.

Any available controllers with assigned IP addresses can be found under the 'Available' tab in the Connections Dialog Box. If the controller is not connected to a DHCP enabled network or the DH command is set to 0, and the controller has not been assigned an IP address, the controller can be found under the 'No IP Address' tab.

For more information on establishing communication to the controller via the GalilTools software, see the GalilTools user manual.

http://www.galilmc.com/support/manuals/galiltools/index.html

Communicating through the Serial Communications Port

Connect the DMC-30000 serial port (labeled SERIAL) to your computer via the Galil CABLE-9PIN-D (RS-232 Cable). This is a straight through serial cable – **NOT** a NULL modem.

Using GalilTools Software for Windows

Registering controllers in the Windows registry is no longer required when using the GalilTools software package. A simple connection dialog box appears when the software is opened that shows all available controllers.

The serial ports are listed as COMn 'communication speed'. (ex COM3 115200). The default USB communication speed on the DMC-30000 is 115200Bps.

For more information on establishing communication to the controller via the GalilTools software, see the GalilTools user manual.

http://www.galilmc.com/support/manuals/galiltools/index.html

Sending Test Commands to the Terminal:

After you connect your terminal, press <return> or the <enter> key on your keyboard. In response to carriage return <return>, the controller responds with a colon, :

Now type:

```
TPA <RETURN>
```

This command directs the controller to return the current position of the A axis. The controller should respond with a number such as

```
:0
```

Step 6. Make Connections to Amplifier and Encoder.

Once you have established communications between the software and the DMC-30000, you are ready to connect the rest of the motion control system. The motion control system typically consists of the controller, an amplifier, and a motor to transform the current from the amplifier into torque for motion. System connection procedures will depend on system components and motor types.

Configuring the DMC-30012, DMC-30016 and DMC-30017 for External Servo Drive

With the controller set to servo mode (MT 1 or -1) to drive an external servo amplifier, the BR command must be set to a -1 (BR -1). This setting will disable the requirement to have the BA, BM and BX or BZ commands executed prior to being able to issue the SH command for that axis (for internal sinusoidally commutated amplifiers) and will configure AO1 for the motor command output.

Connecting to External Amplifiers

If the system is run solely by Galil's integrated amplifiers or drivers, skip this section, the amplifier is already connected to the controller.

Here are the first steps for connecting a motion control system:

Step A. Connect the motor to the amplifier *with no connection to the controller*. Consult the amplifier documentation for instructions regarding proper connections. Connect and turn-on the amplifier power supply. If the amplifiers are operating properly, the motor should stand still even when the amplifiers are powered up.

Step B. Connect the amplifier enable signal.

Before making any connections from the amplifier to the controller, you need to verify that the ground level of the amplifier is either floating or at the same potential as earth.

WARNING: When the amplifier ground is not isolated from the power line or when it has a different potential than that of the computer ground, serious damage may result to the computer, controller and amplifier.

If you are not sure about the potential of the ground levels, connect the two ground signals (amplifier ground and earth) by a 10 kΩ resistor and measure the voltage across the resistor. Only if the voltage is zero, connect the two ground signals directly.

The amplifier enable signal is used by the controller to disable the motor. Use the command, MO, to disable the motor amplifiers - check to insure that the motor amplifiers have been disabled (often this is indicated by an LED on the amplifier).

This signal changes under the following conditions: the watchdog timer activates, the motor-off command, MO, is given, or the OE3 command (Enable Off-On-Error) is given and the position error exceeds the error limit. AEN can be used to disable the amplifier for these conditions.

The AEN signal from the DMC-30000 is 5V active high or high amp enable. In other words, the AEN signal will be high when the controller expects the amplifier to be enabled.

Connecting the Encoders

Step A. Connect the encoders

For stepper motor operation, an encoder is optional.

For servo motor operation, if you have a preferred definition of the forward and reverse directions, make sure that the encoder wiring is consistent with that definition.

The DMC-30000 accepts single-ended or differential encoder feedback with or without an index pulse. The encoder signals are wired to that axis associated 26pin DSub connector found on top of the controller. The signal leads are labeled MA+ (channel A), MB+ (channel B), and MI+. For differential encoders, the complement signals are labeled MA-, MB-, and MI-. For complete pin-out information see in the Appendices.

NOTE: When using pulse and direction encoders, the pulse signal is connected to MA+ and the direction signal is connected to MB+. The controller must be configured for pulse and direction with the command CE. See the command summary for further information on the command CE.

Step B. Verify proper encoder operation.

Start with the A encoder first. Once it is connected, turn the motor shaft and interrogate the position with the instruction TPA <return>. The controller response will vary as the motor is turned.

At this point, if TPA does not vary with encoder rotation, there are three possibilities:

1. The encoder connections are incorrect - check the wiring as necessary.
2. The encoder has failed - using an oscilloscope, observe the encoder signals. Verify that both channels A and B have a peak magnitude between 5 and 12 volts. Note that if only one encoder channel fails, the position reporting varies by one count only. If the encoder failed, replace the encoder. If you cannot observe the encoder signals, try a different encoder.
3. There is a hardware failure in the controller - connect the same encoder to a different axis. If the problem disappears, you may have a hardware failure. Consult the factory for help.

Step 7a. Connect Standard Servo Motors

The following discussion applies to connecting the DMC-30000 controller to standard servo motors:

The motor and the amplifier may be configured in the torque or the velocity mode. In the torque mode, the amplifier gain should be such that a 10 volt signal generates the maximum required current. In the velocity mode, a command signal of 10 volts should run the motor at the maximum required speed. For Galil amplifiers, see A1 – DMC-30012.

Step A. Check the Polarity of the Feedback Loop

It is assumed that the motor and amplifier are connected together and that the encoder is operating correct (Step 6. Make Connections to Amplifier and Encoder.). Before connecting the motor amplifiers to the controller, read the following discussion on setting Error Limits and Torque Limits. Note that this discussion only uses the A axis as an examples.

Step B. Set the Error Limit as a Safety Precaution

Usually, there is uncertainty about the correct polarity of the feedback. The wrong polarity causes the motor to run away from the starting position. Using a terminal program, such as GalilTools, the following parameters can be given to avoid system damage:

Input the commands:

```
ER 2000     Sets error limit on the A axis to be 2000 encoder counts
OE 1        Disables A axis amplifier when excess position error
exists
```

If the motor runs away and creates a position error of 2000 counts, the motor amplifier will be disabled.

NOTE: This function requires the AEN signal to be connected from the controller to the amplifier.

Step C. Set Torque Limit as a Safety Precaution

To limit the maximum voltage signal to your amplifier, the DMC-30000 controller has a torque limit command, TL. This command sets the maximum voltage output of the controller and can be used to avoid excessive torque or speed when initially setting up a servo system.

When operating an amplifier in torque mode, the voltage output of the controller will be directly related to the torque output of the motor. The user is responsible for determining this relationship using the documentation of the motor and amplifier. The torque limit can be set to a value that will limit the motors output torque.

When operating an amplifier in velocity or voltage mode, the voltage output of the controller will be directly related to the velocity of the motor. The user is responsible for determining this relationship using the documentation of the motor and amplifier. The torque limit can be set to a value that will limit the speed of the motor.

For example, the following command will limit the output of the controller to 1 volt on the X axis:

```
TL 1
```

NOTE: Once the correct polarity of the feedback loop has been determined, the torque limit should, in general, be increased to the default value of 9.99. The servo will not operate properly if the torque limit is below the normal operating range. See description of TL in the command reference.

Step D. Connect the Motor Command Output to the Controller

If the system is run solely by Galil's integrated amplifiers or drivers, skip this section, the amplifier is already connected to the controller.

Once the parameters have been set, connect the analog motor command signal (AO1) to the amplifier input.

To test the polarity of the feedback, command a move with the instruction:

```
PR 1000     Position relative 1000 counts
BGA         Begin motion on A axis
```

When the polarity of the feedback is wrong, the motor will attempt to run away. The controller should disable the motor when the position error exceeds 2000 counts. If the motor runs away, the polarity of the loop must be inverted.

Inverting the Loop Polarity

When the polarity of the feedback is incorrect, the user must invert the loop polarity and this may be accomplished by several methods. If you are driving a brush-type DC motor, the simplest way is to invert the two motor wires (typically red and black). For example, switch the M1 and M2 connections going from your amplifier to the motor.

When driving a brushless motor, the polarity reversal may be done with the encoder. If you are using a single-ended encoder, interchange the signal MA+ and MB+. If, on the other hand, you are using a differential encoder, interchange only MA+ and MA-. The loop polarity and encoder polarity can also be affected through software with the MT, and CE commands. For more details on the MT command or the CE command, see the Command Reference section.

To Invert Polarity using Hall-Commutated brushless motors, invert motor phases B & C, exchange Hall A with Hall B, and invert encoder polarity as described above.

Sometimes the feedback polarity is correct (the motor does not attempt to run away) but the direction of motion is reversed with respect to the commanded motion. If this is the case, reverse the motor leads AND the encoder signals.

If the motor moves in the required direction but stops short of the target, it is most likely due to insufficient torque output from the motor command signal AO1. This can be alleviated by reducing system friction on the motors. The instruction:

```
TTA          Tell torque on A
```

reports the level of the output signal. It will show a non-zero value that is below the friction level.

Once you have established that you have closed the loop with the correct polarity, you can move on to the compensation phase (servo system tuning) to adjust the PID filter parameters, KP, KD and KI. It is necessary to accurately tune your servo system to ensure fidelity of position and minimize motion oscillation as described in the next section.

Step 7b. Connect Step Motors

In Stepper Motor operation, the pulse output signal has a 50% duty cycle. Step motors operate open loop and do not require encoder feedback. When a stepper is used, the auxiliary encoder for the corresponding axis is unavailable for an external connection. If an encoder is used for position feedback, connect the encoder to the main encoder input corresponding to that axis. The commanded position of the stepper can be interrogated with RP or TD. The encoder position can be interrogated with TP.

For connecting the stepper motor with the DMC-30016 and DMC-30017, see A2 – DMC-30016 and A3 – DMC-30017 respectively.

If encoders are available on the stepper motor, Galil's Stepper Position Maintenance Mode may be used for automatic monitoring and correction of the stepper position. See Stepper Position Maintenance Mode (SPM) in Chapter 6 Programming for more information.

The frequency of the step motor pulses can be smoothed with the filter parameter, KS. The KS parameter has a range between 0.5 and 128, where 128 implies the largest amount of smoothing. *See Command Reference regarding KS*.

The DMC-30000 profiler commands the step motor amplifier. All DMC-30000 motion commands apply such as PR, PA, VP, CR and JG. The acceleration, deceleration, slew speed and smoothing are also used. Since step motors run open-loop, the PID filter does not function.

To connect step motors with the DMC-30000 you must follow this procedure – If you have a Galil integrated stepper driver (DMC-30016 and DMC-30017) skip Step A, the step and direction lines are already connected to the driver:

Step A. Connect step and direction signals from controller to motor amplifier

The Multi-Function pins are used for the step and direction outputs. See the Multi-Function Pins (MF) section in Chapter 3 Connecting Hardware for more information.

Step B. Configure DMC-30000 for motor type using MT command. You can configure the DMC-30000 for active high or active low pulses. Use the command MT 2 or 2.5 for active low step motor pulses and MT -2 or -2.5 for active high step motor pulses. *See description of the MT command in the Command Reference.*

Note: For the DMC-30012 and DMC-30017, BR-1 must be set prior to setting MT2. BR-1 configures the controller for external amplifier control.

Step 8. Tune the Servo System

Adjusting the tuning parameters is required when using servo motors (standard or sinusoidal commutation). The system compensation provides fast and accurate response and the following section suggests a simple and easy way for compensation. More advanced design methods are available with software design tools from Galil, such as the GalilTools.

The filter has three parameters: the damping, KD; the proportional gain, KP; and the integrator, KI. The parameters should be selected in this order.

To start, set the integrator to zero with the instruction

```
KI 0        Integrator gain
```

and set the proportional gain to a low value, such as

```
KP 1  Proportional gain
KD 100      Derivative gain
```

For more damping, you can increase KD (maximum is 4095.875). Increase gradually and stop after the motor vibrates. A vibration is noticed by audible sound or by interrogation. If you send the command

```
TE A        Tell error
```

a few times, and get varying responses, especially with reversing polarity, it indicates system vibration. When this happens, simply reduce KD by about 20%.

Next you need to increase the value of KP gradually (maximum allowed is 1023.875). You can monitor the improvement in the response with the Tell Error instruction

```
KP 10       Proportion gain
TE A  Tell error
```

As the proportional gain is increased, the error decreases.

Again, the system may vibrate if the gain is too high. In this case, reduce KP by about 20%. Typically, KP should not be greater than KD/4 (only when the amplifier is configured in the current mode).

Finally, to select KI, start with zero value and increase it gradually. The integrator eliminates the position error, resulting in improved accuracy. Therefore, the response to the instruction

```
TE A
```

becomes zero. As KI is increased, its effect is amplified and it may lead to vibrations. If this occurs, simply reduce KI. Repeat tuning for the B, C and D axes.

NOTE: For a more detailed description of the operation of the PID filter and/or servo system theory, see Chapter 10 Theory of Operation and our Manual Tuning Application Note, #3413:
http://www.galilmc.com/support/appnotes/optima/note3413.pdf

Design Examples

Here are a few examples for tuning and using your controller. These examples have remarks next to each command - these remarks must not be included in the actual program.

Example 1 - System Set-up

This example assigns the system filter parameters, error limits and enables the automatic error shut-off.

Instruction	Interpretation

```
KP 10          Implicit Method for setting proportional gain
KPA= 10        Explicit Method for setting proportional gain
KPX= 10        Explicit Method for setting proportional gain
```

Instruction	Interpretation
OE 1	Enable automatic Off on Error function
ER 1000	Set error limit to 1000 counts
KP 10	Set proportional gain

Example 2 - Profiled Move

Rotate the A axis a distance of 10,000 counts at a slew speed of 20,000 counts/sec and an acceleration and deceleration rates of 100,000 counts/s^2. In this example, the motor turns and stops:

Instruction	Interpretation
PR 1000	Distance
SP 20000	Speed
DC 100000	Deceleration
AC 100000	Acceleration
BGA	Start Motion

Example 3 - Absolute Position

Objective: Command motion by specifying the absolute position.

Instruction	Interpretation
DP 2000	Define the current position as 2000
PA 7000	Sets the desired absolute positions to 7000
BGA	Start A motion

Example 4 - Velocity Control

Objective: Drive the motor at a specified speed.

Instruction	Interpretation
JG 10000	Set Jog Speed to 10000 counts/second
AC 100000	Set acceleration to 100000 counts/second^2
DC 50000	Set deceleration to 50000 counts/second^2
BGA	Start motion

after a few seconds, command:

```
JG -40000      New speed and Direction
TVA            Returns A speed
```

This causes velocity changes including direction reversal. The motion can be stopped with the instruction

```
ST             Stop
```

Example 5 - Operation Under Torque Limit

The magnitude of the motor command may be limited independently by the instruction TL.

Instruction	Interpretation
TL 0.2	Set output limit of A axis to 0.2 volts
JG 10000	Set A speed
BGA	Start A motion

In this example, the A motor will probably not move since the output signal will not be sufficient to overcome the friction. If the motion starts, it can be stopped easily by a touch of a finger.

Increase the torque level gradually by instructions such as

Instruction	Interpretation
TL 1.0	Increase torque limit to 1 volt.
TL 9.998	Increase torque limit to maximum, 9.998 volts.

The maximum level of 9.998 volts provides the full output torque.

Example 6 - Interrogation

The values of the parameters may be interrogated. Some examples …

Instruction	Interpretation
KP ?	Return proportional gain
MG _KPA	Return proportional gain

Many other parameters such as KI, KD, FA, can also be interrogated. The command reference denotes all commands which can be interrogated.

Example 7 - Operation in the Buffer Mode

The instructions may be buffered before execution as shown below.

Instruction	Interpretation
PR 600000	Distance
SP 10000	Speed
WT 10000	Wait 10000 milliseconds before reading the next instruction
BGA	Start the motion

Example 8 - Motion Programs with Loops

Motion programs may include conditional jumps as shown below.

Instruction	Interpretation
#A	Label
DP 0	Define current position as zero
v1=1000	Set initial value of v1
#LOOP	Label for loop
PA v1	Move A motor v1 counts

```
BGA                     Start A motion
AMA                     After A motion is complete
WT 500                  Wait 500 ms
TPA                     Tell position A
v1= v1+1000             Increase the value of v1
JP #LOOP,v1<10001       Repeat if v1<10001
EN                      End
```

After the above program is entered, quit the Editor Mode, <cntrl>Q. To start the motion, command:

```
XQ #A                   Execute Program #A
```

Example 9 - Motion Programs with Trippoints

The motion programs may include trippoints as shown below.

Instruction	Interpretation
#B	Label
DP 0	Define initial position
PR 30000	Set target
SP 5000	Set speed
BGA	Start A motion
AP 6000	Wait until position 6000
SP 2000	Change speed
EN	End program

To start the program, command:

```
XQ #B                   Execute Program #B
```

Example 10 - Control Variables

Objective: To show how control variables may be utilized.

Instruction	Interpretation
#A;DP0	Label; Define current position as zero
PR 4000	Initial position
SP 2000	Set speed
BGA	Move A
AMA	Wait until move is complete
WT 500	Wait 500 ms
#B	
v1 = _TPA	Determine distance to zero
PR -v1/2	Command A move 1/2 the distance
BGA	Start A motion
AMA	After A moved
WT 500	Wait 500 ms
MG v1	Report the value of v1

```
JP #C, v1=0              Exit if position=0
JP #B                    Repeat otherwise
#C                       Label #C
EN                       End of Program
```

To start the program, command

```
XQ #A                    Execute Program #A
```

This program moves A to an initial position of 1000 and returns it to zero on increments of half the distance. Note, _TPA is an internal variable which returns the value of the A position.

Chapter 3 Connecting Hardware

Overview

The DMC-30000 provides optoisolated digital inputs for **forward limit**, **reverse limit, home**, and **abort** signals. The controller also has **8 optoisolated uncommitted inputs**, **4 optoisolated outputs**, **2 analog inputs** (0-5V, 12 bit ADC) and **1 uncommitted analog output** (+/-10V, 16-bit DAC).

This chapter describes the inputs and outputs and their proper connection.

Pinout Information can be found in the Connectors for DMC-30000 (Pin-outs) section in the Appendices.

Overview of Optoisolated Inputs

Limit Switch Input

The forward limit switch (FLS) inhibits motion in the forward direction immediately upon activation of the switch. The reverse limit switch (RLS) inhibits motion in the reverse direction immediately upon activation of the switch. If a limit switch is activated during motion, the controller will make a decelerated stop using the deceleration rate previously set with the SD command. The motor will remain on (in a servo state) after the limit switch has been activated and will hold motor position. The controller can be configured to disable the axis upon the activation of a limit switch, see the OE command in the command reference for further detail.

When a forward or reverse limit switch is activated, the current application program that is running in thread zero will be interrupted and the controller will automatically jump to the #LIMSWI subroutine if one exists. This is a subroutine which the user can include in any motion control program and is useful for executing specific instructions upon activation of a limit switch. Automatic Subroutines for Monitoring Conditions are discussed in Chapter 7 Application Programming.

After a limit switch has been activated, further motion in the direction of the limit switch will not be possible until the logic state of the switch returns back to an inactive state. Any attempt at further motion before the logic state has been reset will result in the following error: “22 - Begin not possible due to limit switch” error.

The operands, _LFA and _LRA, contain the state of the forward and reverse limit switch respectively. The value of the operand is either a ‘0’ or ‘1’ corresponding to the logic state of the limit switch. Using a terminal program, the state of a limit switch can be printed to the screen with the command, MG_LFA or MG_LRA. The logic state of the limit switches can also be interrogated with the TS command. For more details on TS see the Command Reference.

Home Switch Input

Homing inputs are designed to provide mechanical reference points for a motion control application. A transition in the state of a Home input alerts the controller that a particular reference point has been reached by a moving part in the motion control system. A reference point can be a point in space or an encoder index pulse.

The Home input detects any transition in the state of the switch and toggles between logic states 0 and 1 at every transition. A transition in the logic state of the Home input will cause the controller to execute a homing routine specified by the user.

There are three homing routines supported by the DMC-30000: Find Edge (FE), Find Index (FI), and Standard Home (HM).

The Find Edge routine is initiated by the command sequence: FEX, BGX. The Find Edge routine will cause the motor to accelerate, and then slew at constant speed until a transition is detected in the logic state of the Home input. The direction of the FE motion is dependent on the state of the home switch. High level causes forward motion. The motor will then decelerate to a stop. The acceleration rate, deceleration rate and slew speed are specified by the user, prior to the movement, using the commands AC, DC, and SP. *When using the FE command, it is recommended that a high deceleration value be used so the motor will decelerate rapidly after sensing the Home switch.*

The Find Index routine is initiated by the command sequence: FIX, BGX. Find Index will cause the motor to accelerate to the user-defined slew speed (SP) at a rate specified by the user with the AC command and slew until the controller senses a change in the index pulse signal from low to high. The motor then decelerates to a stop at the rate previously specified by the user with the DC command and then moves back to the index pulse and speed HV. Although Find Index is an option for homing, it is not dependent upon a transition in the logic state of the Home input, but instead is dependent upon a transition in the level of the index pulse signal.

The Standard Homing routine is initiated by the sequence of commands HMX, BGX. Standard Homing is a combination of Find Edge and Find Index homing. Initiating the standard homing routine will cause the motor to slew until a transition is detected in the logic state of the Home input. The motor will accelerate at the rate specified by the command, AC, up to the slew speed. After detecting the transition in the logic state on the Home Input, the motor will decelerate to a stop at the rate specified by the command, DC. After the motor has decelerated to a stop, it switches direction and approaches the transition point at the speed of HV counts/sec. When the logic state changes again, the motor moves forward (in the direction of increasing encoder count) at the same speed, until the controller senses the index pulse. After detection, it decelerates to a stop, moves back to the index, and defines this position as 0. The logic state of the Home input can be interrogated with the command MG_HMX. This command returns a 0 or 1 if the logic state is low or high, respectively. The state of the Home input can also be interrogated indirectly with the TS command.

For examples and further information about Homing, see command HM, FI, FE of the Command Reference and the section entitled Homing in the Programming Motion Section of this manual.

Abort Input

The function of the Abort input is to immediately stop the controller upon transition of the logic state.

NOTE: The response of the abort input is significantly different from the response of an activated limit switch. When the abort input is activated, the controller stops generating motion commands immediately, whereas the limit switch response causes the controller to make a decelerated stop.

NOTE: The effect of an Abort input is dependent on the state of the off-on-error function (OE Command). If the Off-On-Error function is enabled the motor will be turned off when the abort signal is generated. This could cause the motor to 'coast' to a stop since it is no longer under servo control. If the Off-On-Error function is disabled, the motor will decelerate to a stop as fast as mechanically possible and the motor will remain in a servo state.

All motion programs that are currently running are terminated when a transition in the Abort input is detected. This can be configured with the CN command. For information see the Command Reference, OE and CN.

ELO (Electronic Lock-Out) Input

Used in conjunction with Galil amplifiers, this input allows the user the shutdown the amplifier at a hardware level. For more detailed information on how specific Galil amplifiers behave when the ELO is triggered, see individual sections in the Appendices.

Reset Input/Reset Button

When the Reset line is triggered the controller will be reset. The reset line and reset button will not master reset the controller unless the MRST jumper is installed during a controller reset.

Uncommitted Digital Inputs

The DMC-30000 has 8 optoisolated inputs. These inputs can be read individually using the function @ IN[x] where x specifies the input number (1 thru 8). These inputs are uncommitted and can allow the user to create conditional statements related to events external to the controller. For example, the user may wish to have the motor move 1000 counts in the positive direction when the logic state of DI1 goes high.

Digital Input 1 can be used has a high speed position latch, see High Speed Position Capture (The Latch Function) for more information.

This can be accomplished by connecting a voltage in the range of +5V to +28V into INC of the input circuitry from a separate power supply.

Optoisolated Input Electrical Information

Electrical Specifications

Input Common (INC) and Digital Input Max Voltage	28 VDC
Input Common (INC) and Digital Input Min Voltage	0 VDC
Limit Common (LSC) and Limit/Home Input Max Voltage	28 VDC
Input Command (INC) and Limit/Home Input Max Min Voltage	0 VDC
Minimum current to turn on Inputs	1.2 mA
Minimum current to turn off Inputs once activated (hysteresis)	0.5 mA
Internal Resistance of Inputs (INC and LSC to Inputs)	2.2 kΩ

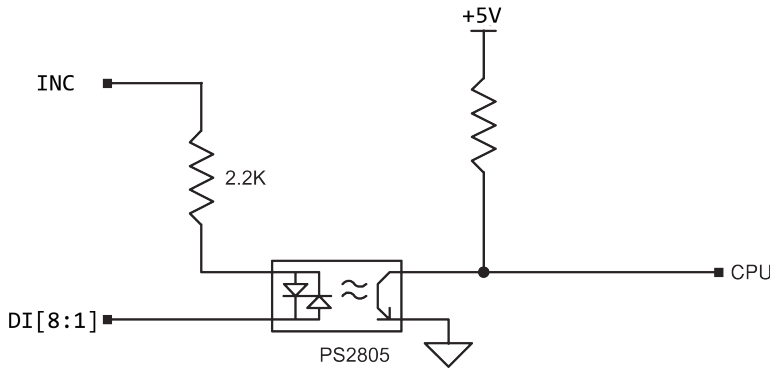


Figure 3.1: Uncommitted Digital inputs on the DMC-30000

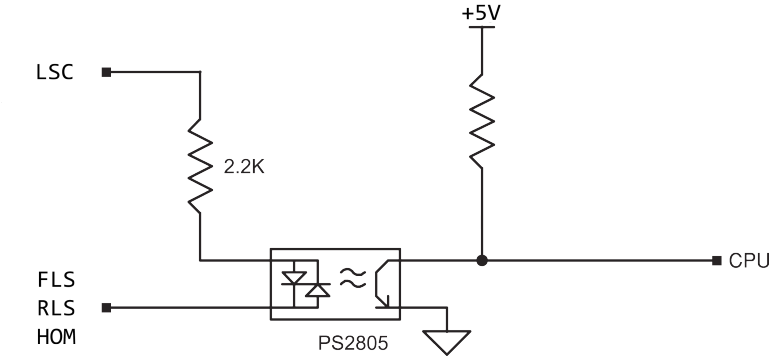


Figure 3.2: Limit Switch Inputs on the DMC-30000

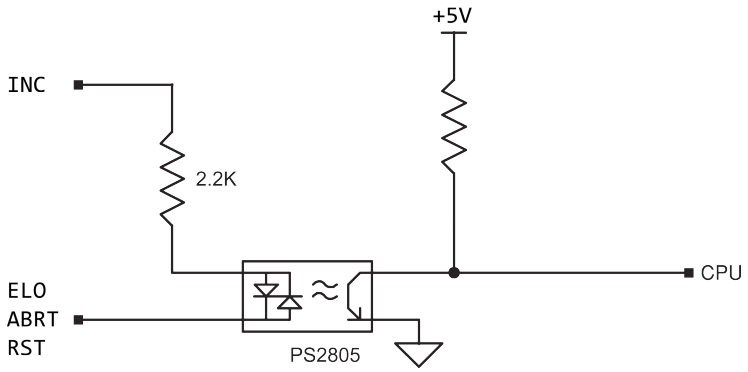


Figure 3.3: ELO, Abort and Reset Inputs on the DMC-30000

Wiring the Optoisolated Digital Inputs

All inputs can be used as active high or low - If you are using an isolated power supply you can connect the positive voltage of the supply (+Vs) to INC or supply the isolated ground to INC. Connecting +Vs to INC will configure the inputs for active low. Connecting the isolated ground to INC will configure the inputs for active high. If there is not an isolated supply available, the Galil 5V and GND may be used. It is recommended to use an isolated supply for the optoisolated inputs.

To take full advantage of optoisolation, an isolated power supply should be used to provide the voltage at the input common connection. When using an isolated power supply, do not connect the ground of the isolated power to the ground of the controller. A power supply in the voltage range between 5 to 28 Volts may be applied directly (see Figure 3.4).

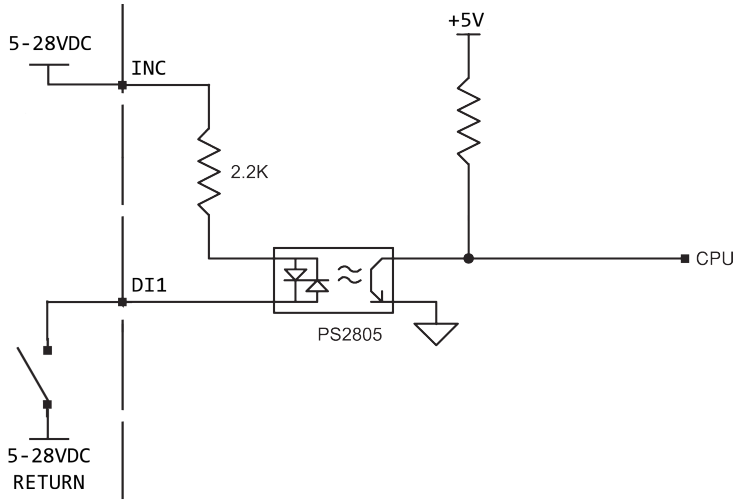


Figure 3.4: Digital Input Wiring for +Vs to INC

The optoisolated inputs are configured into groups. For example, the general inputs, DI[8:1] (inputs 1-8), the ABRT (abort) input and RST (reset) and ELO (electronic lock-out) inputs are one group. The Limit and Home Switches are in another group.

The optoisolated inputs are connected in the following groups:

Group	Common Signal
DI1-DI8, ABRT, RST, ELO	INC
FLS,RLS,HOM	LSC

Table 3.1: INC and LSC information

Using Voltages > 28 VDC

For voltages greater than 28 Volts, a resistor, R, is needed in series with the input such that:

1 mA < Vs / (R + 2.2KΩ) < 11 mA

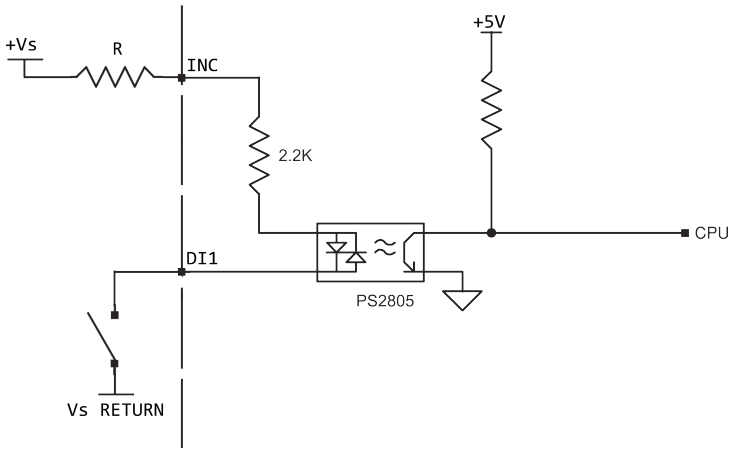


Figure 3.5: Wiring inputs for > 28VDC

Bypassing the Optoisolation

If no isolation is needed, the internal 5 Volt supply may be used to power the switches. This can be done by connecting LSC or INC to 5V.

To close the circuit, wire the desired input to any ground (GND) pin on the controller.

Optoisolated Outputs

The DMC-30000 has several different options for the uncommitted digital outputs (labeled as DO). The default outputs are 4mA sinking which are ideal for interfacing to TTL level devices. Additional options include 25mA sinking (lower power sinking, LSNK), 25mA sourcing (low power sourcing, LSRC), 500mA sourcing (high power sourcing, HSRC), and 500mA sinking outputs (high power sinking, HSNK). Please refer to your part number to determine which option you have.

The DMC-30000 has only has a single bank (Bank 0) of 4 optoisolated outputs, powered through the Output PWR and Output GND pins located on J5 - I/O 44 pin HD D-Sub Connector (Female). Please see the Connectors for DMC-30000 (Pin-outs) in the Appendix for pin-outs.

Wiring diagrams, electrical specifications, and details for each output type are provided below.

Brake Output

When using the brake outputs, it is recommended to order the controller with 500mA sourcing output option (HSRC).

Outputs 1 is the brake output.

The BW command sets the delay between when the brake is turned on and when the amp is turned off. When the controller goes into a motor-off (MO) state, this is the time (in samples) between when the brake digital output changes state and when the amp enable digital output changes state. The brake is actuated immediately upon MO and the delay is to account for the time it takes for the brake to engage mechanically once it is energized electrically. The brake is released immediately upon SH.

See the BW command in the DMC-30000 Command Reference for more information.

Standard 4mA Sinking Optoisolated Outputs

Description

The default outputs of the DMC-30000 are capable of 4mA and are configured as sinking outputs. The voltage range for the outputs is 5-24 VDC. These outputs should not be used to drive inductive loads directly.

Electrical Specifications

Output PWR Max Voltage	24 VDC
Output PWR Min Voltage	5 VDC
ON Voltage (No Load, Output PWR= 5VDC)	0.1 VDC
Max Drive Current per Output	4mA – Sinking

Wiring the Standard 4mA outputs

With this configuration, the output power supply will be connected to Output PWR (labeled OPB) and the power supply return will be connected to Output GND (labeled OPA). Note that the load is wired between Output PWR and DO. The wiring diagram for Bank 0 is shown in Error: Reference source not found. Refer to Connectors for DMC-30000 (Pin-outs) in the Appendix for pin-out information.

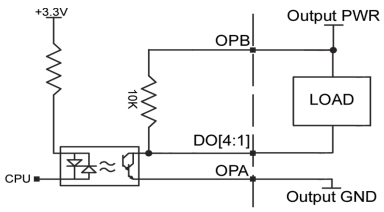


Figure 3.6: 4mA sinking wiring diagram for Bank 0, DO [4:1]

25mA Sinking Optoisolated Outputs (LSNK)

Description

The 25mA sinking option, refereed to as lower power sinking (LSNK), are capable of sinking up to 25mA per output. The voltage range for the outputs is 5-24 VDC. These outputs should not be used to drive inductive loads directly.

Electrical Specifications

Output PWR Max Voltage	24 VDC
Output PWR Min Voltage	5 VDC
ON Voltage (No Load, Output PWR= 5 VDC)	1.2 VDC
Max Drive Current per Output	25mA, Sinking

Wiring the 25mA Sinking Outputs

The 25mA sinking outputs the load is wired in the same fashion as the 4mA sinking outputs: The output power supply will be connected to Output PWR (labeled OPB) and the power supply return will be connected to Output GND (labeled OPA). Note that the load is wired between Output PWR and DO. The wiring diagram for Bank 0 is shown in Figure 3.7. Refer to Connectors for DMC-30000 (Pin-outs) in the Appendix for pin-out information.

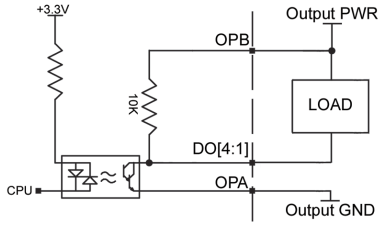


Figure 3.7: 25mA sinking wiring diagram for Bank 0, DO[4:1]

25mA Sourcing Optoisolated Outputs (LSRC)

Description

The 25mA sourcing option, refereed to as low power sourcing (LSRC), are capable of sourcing up to 25mA per output. The voltage range for the outputs is 5-24 VDC. These outputs should not be used to drive inductive loads directly.

Electrical Specifications

Output PWR Max Voltage	24 VDC
Output PWR Min Voltage	5 VDC
Max Drive Current per Output	25mA, Sourcing

Wiring the 25mA Sourcing Outputs

With this configuration, the output power supply will be connected to Output PWR (labeled OPA) and the power supply return will be connected to Output GND (labeled OPB). Note that the load is wired between DO and Output GND. The wiring diagram for Bank 0 is shown in Figure 3.8 . Refer to Connectors for DMC-30000 (Pin-outs) in the Appendix for pin-out information.

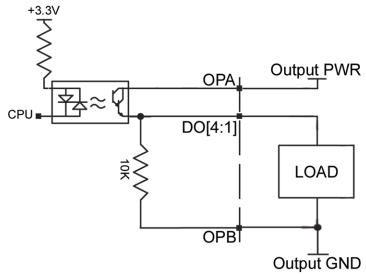


Figure 3.8: 25mA sourcing wiring diagram for Bank 0, DO[4:1]

500mA Sourcing Optoisolated Outputs (HSRC)

Description

The 500mA sourcing option, refereed to as high power sourcing (HSRC), is capable of sourcing up to 500mA per output and up to 1.5 A per **bank**. The voltage range for the outputs is 12-24 VDC. These outputs are capable of driving inductive loads such as solenoids or relays. The outputs are configured for hi-side (sourcing).

Electrical Specifications

Output PWR Max Voltage	24 VDC
Output PWR Min Voltage	12 VDC
Max Drive Current per Output	0.5 A (not to exceed 1.5 A for all 4 outputs)

Wiring the 500mA Sourcing Optoisolated Outputs

With this configuration, the output power supply will be connected to Output PWR (labeled OPA) and the power supply return will be connected to Output GND (labeled OPB). Note that the load is wired between DO and Output GND. The wiring diagram for Bank 0 is shown in Figure 3.9. Refer to Connectors for DMC-30000 (Pin-outs) in the Appendix for pin-out information.

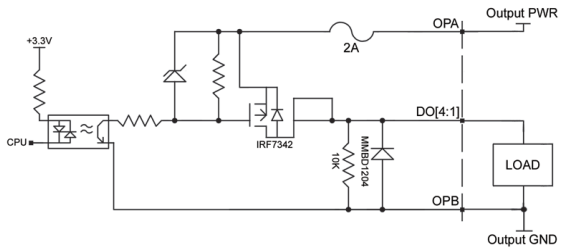


Figure 3.9: 500mA sourcing wiring diagrams for Bank 0, DO[4:1]

500mA Sinking Optoisolated Outputs (HSNK)

Description

The 500mA sinking option, refereed to as high power sinking (HSNK), is capable of sinking up to 500mA per output and up to 1.5 A per **bank**. The voltage range for the outputs is 12-24 VDC. These outputs are capable of driving inductive loads such as solenoids or relays. The outputs are configured for low-side (sinking).

Electrical Specifications

Output PWR Max Voltage	24 VDC
Output PWR Min Voltage	12 VDC
Max Sink Current per Output	0.5 A (not to exceed 1.5 A for all 4 outputs)

Wiring the 500mA Sinking Optoisolated Outputs

With this configuration, the output power supply will be connected to Output PWR (labeled OPB) and the power supply return will be connected to Output GND (labeled OPA). Note that the load is wired between Output PWR and DO. The wiring diagram for Bank 0 is shown in Figure 3.10. Refer to Connectors for DMC-30000 (Pin-outs) in the Appendix for pin-out information.

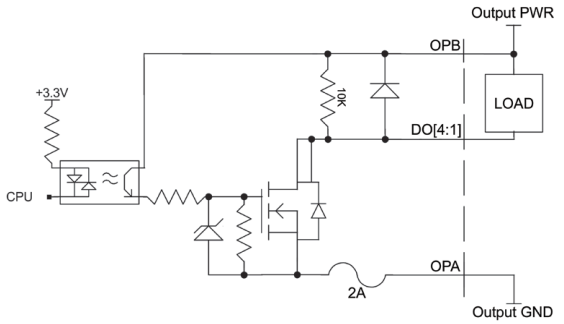


Figure 3.10: 500mA sinking wiring diagram for Bank 0, DO[4:1]

Feedback Inputs and Multi-Function (MF) Pins

Feedback Options

There are many different options for feedback with the DMC-30000 series controllers. The indicates which feedback options are available with each configuration, and the inputs for those feedback options.

DMC-30000 Feedback Options				
	DMC-3001x	DMC-3001x-SER	DMC-3101x	DMC-3101x-SER
Main Digital Encoder	MA/MB	MA/MB	MA/MB	MA/MB
Aux Digital Encoder	AA/AB	AA/AB	AA/AB	AA/AB
Analog Feedback (0-5V)	AI1	AI1	AI1 (AQ)	AI1 (AQ)
Analog Feedback (16 bit configurable +/-10V)	-	-	AI1 (AQ)	AI1 (AQ)
SSI/BiSS Channel 0	-	MF0 – Main	-	MF0 – Main
SSI/BiSS Channel 1	-	MF1 – Aux	-	MF1 – Aux
Sin/Cos Encoder	-	-	MA/MB	MA/MB

Table 3.2: Feedback options for DMC-30000 series controllers

- MA/MB are the Main Encoder inputs
- AA/AB are the Aux Encoder Inputs
- AI 1 is Analog Input 1
- MF 0 is Multi-Funtion Input 0
- MF 1 is Multi-Function Input 1

Main Encoder Inputs

The main encoder inputs can be configured for quadrature (default) or pulse and direction inputs. This configuration is set through the CE command. The encoder connections are found on the 15 pin HD D-sub Encoder connectors and are labeled MA+, MA-, MB+, MB-. The '-' (negative) inputs are the differential inputs to the encoder inputs; if the encoder is a single ended 5V encoder, then the negative input should be left floating (except for the DMC-31000). If the encoder is a single ended and outputs a 0-12V signal then the negative input should be tied to the 5V line on the DMC-30000.

When the encoders are setup as step and direction inputs the MA channel will be the step or pulse input, and the MB channel will be the direction input.

The encoder inputs can be ordered with 120Ω termination resistors installed. See TRES – Encoder Termination Resistors in the Appendix for more information.

Electrical Specifications

Maximum Voltage 12 VDC

Minimum Voltage -12 VDC

Maximum Frequency (Quadrature) 15 MHz

'+' inputs are internally pulled-up to 5V through a 4.7 kΩ resistor

'-' inputs are internally biased to ~1.3V

pulled up to 5V through a 7.1 kΩ resistor

pulled down to GND through a 2.5 kΩ resistor

The Auxiliary Encoder Inputs

The auxiliary encoder inputs can be used for general use. The controller has one auxiliary encoder which consists of two inputs, channel A and channel B. The auxiliary encoder inputs are mapped to the inputs 81 and 82. The Aux encoder inputs are not available when the controller is configured for step and direction outputs (stepper).

Each input from the auxiliary encoder is a differential line receiver and can accept voltage levels between +/- 12 volts. The inputs have been configured to accept TTL level signals. To connect TTL signals, simply connect the signal to the + input and leave the - input disconnected. For other signal levels, the '-' input should be connected to a voltage that is ~½ of the full voltage range (for example, connect the '-' input to the 5 volts on the Galil if the signal is 0 - 12V logic).

Electrical Specifications

Maximum Voltage 12 VDC

Minimum Voltage -12 VDC

'+' inputs are internally pulled-up to 5V through a 4.7kΩ resistor

'-' inputs are internally biased to ~1.3V

pulled up to 5V through a 7.1kΩ resistor

pulled down to GND through a 2.5kΩ resistor

Multi-Function Pins (MF)

Multi-Functional Pins (MFn+/-)

The Multi-Functional Pins on the DMC-30000 have different functionalities dependent upon how the controller was ordered and how the controller is setup by the user. If the controller is ordered with -SER (serial encoder interface), then the MF pins can be used to interface to a serial encoder (BiSS and SSI). MF1 and MF2 are only used for the Main serial encoder input, MF2 and MF3 are used for the Aux serial encoder input. See the SI and SS commands in the command reference for more detail.

When the controller is setup for stepper motor operation, the MF 2 and MF4 pins are used for step and direction respectively.

Single Description for Multi-Functional Pins			
Label	**Pin #**	**MT +/-2 or +/-2.5**	**-SER option with BiSS or SSI Enabled**
MF1 +	35	No Connect	Main Axis Data + (D0+ or SLO+)
MF1 -	5	No Connect	Main Axis Data - (D0- or SLO-)
MF2 +	6	STEP +	Main Axis Clock + (C0+ or MA+)
MF2 -	21	STEP -	Main Axis Clock - (C0- or MA-)
MF3 +	22	No Connect	Aux Axis Data + (D1+ or SLO+)
MF3 -	36	No Connect	Aux Axis Data - (D1- or SLO-)
MF4 +	37	DIR +	Aux Axis Clock + (C1+ or MA+)
MF4 -	7	DIR -	Aux Axis Clock - (C1- or MA-)

Electrical Specifications (MF2, MF4)

Output Voltage 0 – 3.3 VDC

Current Output 4 mA Sink/Source

Electrical Specifications (MF1, MF3)

Maximum Input Voltage	5 VDC
Minimum Input Voltage	0 VDC

TTL Outputs

Step and Direction Outputs

The Multi-Function Pins (MF) are used for the external step and direction outputs.

Output Compare

The output compare signal is TTL and is available on the I/O D-Sub connector as CMP. Output compare is controlled by the position of the main encoder input on the controller. The output can be programmed to produce an active low pulse (510 nsec) based on an incremental encoder value or to activate once when an axis position has been passed. When setup for a one shot, the output will stay low until the OC command is called again. For further information, see the command OC in the Command Reference.

Electrical Specifications

Output Voltage	0 – 5 VDC
Current Output	20 mA Sink/Source

Error Output

The controller provides a TTL signal, ERR, to indicate a controller error condition. When an error condition occurs, the ERR signal will go low and the controller LED will go on. An error occurs because of one of the following conditions:

1. At least one axis has a position error greater than the error limit. The error limit is set by using the command ER.
2. The reset line on the controller is held low or is being affected by noise.
3. There is a failure on the controller and the processor is resetting itself.
4. There is a failure with the output IC which drives the error signal.

The ERR signal is found on the I/O (A-D) D-Sub connector.

For additional information see Error Light (Red LED) in Chapter 9 Troubleshooting.

Electrical Specifications

Output Voltage	0 – 5 VDC
Current Output	20 mA Sink/Source

Analog Inputs

DMC-30000

The DMC-30000 has two 0-5V analog inputs. The inputs are decoded by a 12-bit A/D decoder giving a voltage resolution of approximately 1.22mV. The analog inputs are specified as AN[x] where x is a number 1 or 2.

The analog inputs can be set to a differential mode where analog input 2 is the differential input to analog input 1.

Electrical Specifications

Maximum Voltage	5V
Minimum Voltage	0V
Resolution	12 bit
Input Impedance	100 kΩ

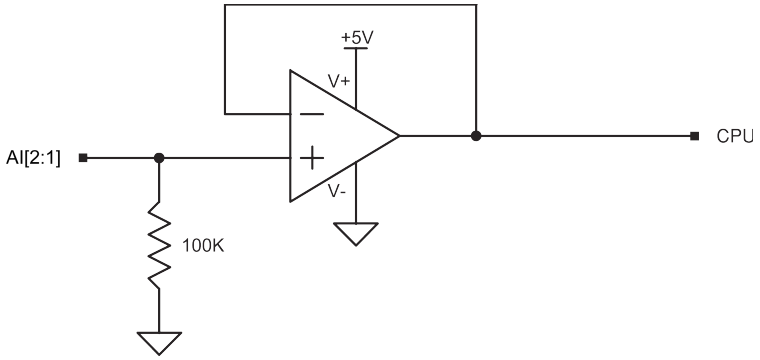


Figure 3.11: Analog Inputs for the DMC-30000

DMC-31000

The DMC-31000 has two analog inputs configured for the range between -10V and 10V. The inputs are decoded by a 16-bit A/D decoder giving a voltage resolution of approximately .0003V. The analog inputs are specified as AN[x] where x is a number 1 thru 2.

AQ settings

With the DMC-31000, the analog inputs can be set to a range of +/-10V, +/-5V, 0-5V or 0-10V, this allows for increased resolution when the full +/-10V is not required. The inputs can also be set into a differential mode where analog input 2 can be set to the negative differential input for analog input 1. See the AQ command in the command reference for more information.

Electrical Specifications

Resolution		16 bit
Input Impedance –		
	Unipolar (0-5V, 0-10V)	42kΩ
	Bipolar (+/-5V, +/-10V)	31kΩ

External Amplifier Interface

Electrical Specifications

Max Amplifier Enable Voltage	5V	
Max Amplifier Enable Current	sink/source	20 mA

Overview

The motor command output on the DMC-30010 and DMC-30011 controllers is labeled AO1.

The DMC-30000 command voltage ranges between +/-10V and is output on the motor command line (AO1). This signal, along with GND, provides the input to the motor amplifiers. The amplifiers must be sized to drive the motors and load. For best performance, the amplifiers should be configured for a torque (current) mode of operation with no additional compensation. The gain should be set such that a 10 volt input results in the maximum required current.

The DMC-30000 also provides an amplifier enable signal - AEN. This signal changes under the following conditions: the motor-off command, MO, is given, the watchdog timer activates, or the OE command (Enable Off-On-Error) is set and the position error exceeds the error limit or a limit switch is reached (see OE command in the Command Reference for more information).

The amplifier enable signal is 5V active high amp enable (HAEN). In other words, the AEN signal will be high when the controller expects the amplifier to be enabled.

NOTE: Many amplifiers designate the enable input as 'inhibit'.

Settings for DMC-30012 and DMC-30017

External Stepper Drivers

With the controller set to stepper mode (MT -2, 2, 2.5 or -2.5), the step and direction outputs are found on the I/O connector list as MF2 and MF4. Details and pinout information can be found in the Multi-Function Pins (MF) section of Chapter 3.

External Servo Amplifiers

With the controller set to servo mode (MT 1 or -1) to drive an external servo amplifier, the BR command must be set to a -1. This setting will disable the requirement to have the BA, BM and BX or BZ commands executed prior to being able to issue the SH command for that axis.

Analog Outputs/Motor Command Output

Electrical Specifications

Voltage Range	+/-10V
Output Impedance	500 Ω
Resolution	16 bit

Overview

The DMC-30000 has 2 analog outputs, AO 1 and AO 2. AO 2 is always used as a general purpose output. AO 1 is used as the motor command output when using the DMC-30000 to interface to an external amplifier.

If the controller is set to control stepper motors (MT -2, 2, -2.5 or 2.5), then AO1 will be +/-10V depending on MT setting and the commanded speed to the motor.

See the AO command in the DMC-30000 Command Reference for more information.

AO1 with the DMC-30012 and DMC-30017

AO1 as External Motor Command

To configure the DMC-30012 and DMC-30017 to interface to an external servo amplifier, set MT1 or -1 and set BR-1. In this mode AO1 will be the motor command output to the external drive and will not be available as a general purpose output.

AO1 as a General Purpose Output

With a DMC-30012 and DMC-30017 AO1 can be used as a general purpose output when the controller is set to drive a brushless servo motor with the internal amplifier. AO 1,n where n is a number from -10 to 10 will set analog output 1 with a DMC-30012 and DMC-30017.

The analog output can be set with the AO command once MT is set to 1 or -1 and the BA command is set for the A axis (BA A).

Chapter 4 Software Tools and Communication

Introduction

The default configuration DMC-30000 has one RS-232 port and two Ethernet ports. The RS-232 port baud rate defaults to 115200 bps and can be configured for 19200 bps via jumpers on the side of the controller. The Ethernet ports are 100BASE-T connections that auto-negotiate half or full duplex.

The GalilTools software package is available for PC computers running Microsoft Windows or Linux to communicate with the DMC-30000 controller. This software package has been developed to operate under Windows and Linux, and include all the necessary drivers to communicate to the controller. In addition, GalilTools includes a software development communication library which allows users to create their own application interfaces using programming environments such as C, C++, Visual Basic, and LabVIEW.

The following sections in this chapter are a description of the communications protocol, and a brief introduction to the software tools and communication techniques used by Galil. At the application level, GalilTools is the basic programs that the majority of users will need to communicate with the controller, to perform basic setup, and to develop application code (.dmc programs) that is downloaded to the controller. At the Galil API level, the GalilTools Communication Library is available for users who wish to develop their own custom application programs to communicate to the controller. Custom application programs can utilize API function calls directly to our DLL's. At the driver level, we provide fundamental hardware interface information for users who desire to create their own drivers.

Controller Response to Commands

Most DMC-30000 instructions are represented by two characters followed by the appropriate parameters. Each instruction must be terminated by a carriage return. Multiple commands may be concatenated by inserting a semicolon between each command.

After the instruction is decoded, the DMC-30000 returns a response to the port from which the command was generated. If the instruction was valid, the controller returns a colon (:) or the controller will respond with a question mark (?) if the instruction was not valid. For example, the controller will respond to commands which are sent via the RS-232 port back through the RS-232 port, and to commands which are sent via the Ethernet port back through the Ethernet port.

For instructions that return data, such as Tell Position (TP), the DMC-30000 will return the data followed by a carriage return, line feed and : .

It is good practice to check for : after each command is sent to prevent errors. An echo function is provided to enable associating the DMC-30000 response with the data sent. The echo is enabled by sending the command EO 1 to the controller.

Unsolicited Messages Generated by Controller

When the controller is executing a program, it may generate responses which will be sent via the RS-232 port or Ethernet handles. This response could be generated as a result of messages using the MG command OR as a result of a command error. These responses are known as unsolicited messages since they are not generated as the direct response to a command.

Messages can be directed to a specific port using the specific Port arguments – see the MG and CF commands in the Command Reference. If the port is not explicitly given or the default is not changed with the CF command, unsolicited messages will be sent to the default port. The default port is the serial port. When communicating via an Ethernet connection, the unsolicited messages must be sent through a handle that is not the main communication handle from the host. The GalilTools software automatically establishes this second communication handle.

The controller has a special command, CW, which can affect the format of unsolicited messages. This command is used by Galil Software to differentiate response from the command line and unsolicited messages. The command, CW1 causes the controller to set the high bit of ASCII characters to 1 of all unsolicited characters. This may cause characters to appear garbled to some terminals. This function can be disabled by issuing the command, CW2. For more information, see the CW command in the Command Reference.

RS-232 Port

Cable requirements

The RS-232 port on the DMC-30000 requires a straight through serial cable. The pinout for this cable is indicated below:

RS232 - Main Port {P1} DATATERM

1 No Connect
2 Transmit Data - output
3 Receive Data - input
4 No Connect
5 Ground
6 No Connect
7 Clear To Send - input
8 Request To Send - output
9 No connect

Configuration

The GalilTools software will automatically configure your PC for 8-bit data, one start-bit, one stop-bit, full duplex and no parity. The baud rate for the RS-232 communication can be selected by setting the proper switch configuration on the front panel according to the table below.

Baud Rate Selection

JP1 JUMPER SETTINGS	
19.2	**BAUD RATE**
ON	19200
OFF (recommended)	115200

Handshaking

The RS-232 main port is set for hardware handshaking. Hardware Handshaking uses the RTS and CTS lines. The CTS line will go high whenever the DMC-30000 is not ready to receive additional characters. The RTS line will inhibit the DMC-30000 from sending additional characters. Note, the RTS line goes high for inhibit.

RS-422 Configuration

The DMC-30000 can be ordered with the auxiliary port configured for RS-422 communication. RS-422 communication is a differentially driven serial communication protocol that should be used when long distance serial communication is required in an application.

For more information see RS-422 – Serial Port Serial Communication in the in Appendix.

Ethernet Configuration

Communication Protocols

The Ethernet is a local area network through which information is transferred in units known as packets. Communication protocols are necessary to dictate how these packets are sent and received. The DMC-30000 supports two industry standard protocols, TCP/IP and UDP/IP. The controller will automatically respond in the format in which it is contacted.

TCP/IP is a "connection" protocol. The master, or client, connects to the slave, or server, through a series of packet handshakes in order to begin communicating. Each packet sent is acknowledged when received. If no acknowledgment is received, the information is assumed lost and is resent.

Unlike TCP/IP, UDP/IP does not require a "connection". If information is lost, the controller does not return a colon or question mark. Because UDP does not provide for lost information, the sender must re-send the packet.

It is recommended that the motion control network containing the controller and any other related devices be placed on a “closed” network. If this recommendation is followed, UDP/IP communication to the controller may be utilized instead of a TCP connection. With UDP there is less overhead, resulting in higher throughput. Also, there is no need to reconnect to the controller with a UDP connection. Because handshaking is built into the Galil communication protocol through the use of colon or question mark responses to commands sent to the controller, the TCP handshaking is not required.

Packets must be limited to 512 data bytes (including UDP/TCP IP Header) or less. Larger packets could cause the controller to lose communication.

NOTE: In order not to lose information in transit, the user must wait for the controller's response before sending the next packet.

Addressing

There are three levels of addresses that define Ethernet devices. The first is the MAC or hardware address. This is a unique and permanent 6 byte number. No other device will have the same MAC address. The DMC-30000 MAC address is set by the factory and the last two bytes of the address are the serial number of the board. To find the Ethernet MAC address for a DMC-30000 unit, use the TH command. A sample is shown here with a unit that has a serial number of 11:

Sample MAC Ethernet Address: 00-50-4C-40-00-0B

The second level of addressing is the IP address. This is a 32-bit (or 4 byte) number that usually looks like this: 192.168.15.1. The IP address is constrained by each local network and must be assigned locally. Assigning an IP address to the DMC-30000 controller can be done in a number of ways.

The first method for setting the IP address is using a DHCP server. The DH command controls whether the DMC-30000 controller will get an IP address from the DHCP server. If the unit is set to DH1 (default) and there is a DHCP server on the network, the controller will be dynamically assigned an IP address from the server. Setting the board to DH0 will prevent the controller from being assigned an IP address from the server.

The second method to assign an IP address is to use the BOOT-P utility via the Ethernet connection. The BOOT-P functionality is only enabled when DH is set to 0. Either a BOOT-P server on the internal network or the Galil software may be used. When opening the Galil Software, it will respond with a list of all DMC-30000’s and other

controllers on the network that do not currently have IP addresses. The user must select the board and the software will assign the specified IP address to it. This address will be burned into the controller (BN) internally to save the IP address to the non-volatile memory.

NOTE: if multiple boards are on the network – use the serial numbers to differentiate them.

CAUTION: Be sure that there is only one BOOT-P or DHCP server running. If your network has DHCP or BOOT-P running, it may automatically assign an IP address to the DMC-30000 controller upon linking it to the network. In order to ensure that the IP address is correct, please contact your system administrator before connecting the I/O board to the Ethernet network.

The third method for setting an IP address is to send the IA command through the RS-232 port. (Note: The IA command is only valid if DH0 is set). The IP address may be entered as a 4 byte number delimited by commas (industry standard uses periods) or a signed 32 bit number (e.g. IA 124,51,29,31 or IA 2083724575). Type in BN to save the IP address to the DMC-30000 non-volatile memory.

NOTE: Galil strongly recommends that the IP address selected is not one that can be accessed across the Gateway. The Gateway is an application that controls communication between an internal network and the outside world.

The third level of Ethernet addressing is the UDP or TCP port number. The Galil board does not require a specific port number. The port number is established by the client or master each time it connects to the DMC-30000 board. Typical port numbers for applications are:

Port 23: Telnet
Port 502: Modbus

Communicating with Multiple Devices

The DMC-30000 is capable of supporting multiple masters and slaves. The masters may be multiple PC's that send commands to the controller. The slaves are typically peripheral I/O devices that receive commands from the controller.

NOTE: The term "Master" is equivalent to the internet "client". The term "Slave" is equivalent to the internet "server".

An Ethernet handle is a communication resource within a device. The DMC-30000 can have a maximum of 6 Ethernet handles open at any time. When using TCP/IP, each master or slave uses an individual Ethernet handle. In UDP/IP, one handle may be used for all the masters, but each slave uses one. (Pings and ARPs do not occupy handles.) If all 6 handles are in use and a 7th master tries to connect, it will be sent a "reset packet" that generates the appropriate error in its windows application.

NOTE: There are a number of ways to reset the controller. Hardware reset (push reset button or power down controller) and software resets (through Ethernet or RS-232 by entering RS).

When the Galil controller acts as the master, the IH command is used to assign handles and connect to its slaves. The IP address may be entered as a 4 byte number separated with commas (industry standard uses periods) or as a signed 32 bit number. A port number may also be specified, but if it is not, it will default to 1000. The protocol (TCP/IP or UDP/IP) to use must also be designated at this time. Otherwise, the controller will not connect to the slave. (Ex. IHB=151,25,255,9<179>2 This will open handle #2 and connect to the IP address 151.25.255.9, port 179, using TCP/IP)

Which devices receive what information from the controller depends on a number of things. If a device queries the controller, it will receive the response unless it explicitly tells the controller to send it to another device. If the command that generates a response is part of a downloaded program, the response will route to whichever port is specified as the default (unless explicitly told to go to another port with the CF command). To designate a specific destination for the information, add {Eh} to the end of the command. (Ex. MG{EC}"Hello" will send the message "Hello" to handle #3. TP,,?{EF} will send the z axis position to handle #6.)

Multicasting

A multicast may only be used in UDP/IP and is similar to a broadcast (where everyone on the network gets the information) but specific to a group. In other words, all devices within a specified group will receive the information that is sent in a multicast. There can be many multicast groups on a network and are differentiated by their multicast IP address. To communicate with all the devices in a specific multicast group, the information can be sent to the multicast IP address rather than to each individual device IP address. All Galil controllers belong to a default multicast address of 239.255.19.56. The controller's multicast IP address can be changed by using the IA> u command.

Using Third Party Software

Galil supports DHCP, ARP, BOOT-P, and Ping which are utilities for establishing Ethernet connections. DHCP is a protocol used by networked devices (clients) to obtain the parameters necessary for operation in an Internet Protocol network. ARP is an application that determines the Ethernet (hardware) address of a device at a specific IP address. BOOT-P is an application that determines which devices on the network do not have an IP address and assigns the IP address you have chosen to it. Ping is used to check the communication between the device at a specific IP address and the host computer.

The DMC-30000 can communicate with a host computer through any application that can send TCP/IP or UDP/IP packets. A good example of this is Telnet, a utility that comes with most Windows systems.

Modbus

An additional protocol layer is available for speaking to I/O devices. Modbus is an RS-485 protocol that packages information in binary packets that are sent as part of a TCP/IP packet. In this protocol, each slave has a 1 byte slave address. The DMC-30000 can use a specific slave address or default to the handle number. The port number for Modbus is 502.

The Modbus protocol has a set of commands called function codes. The DMC-30000 supports the 10 major function codes:

Function Code	Definition
01	Read Coil Status (Read Bits)
02	Read Input Status (Read Bits)
03	Read Holding Registers (Read Words)
04	Read Input Registers (Read Words)
05	Force Single Coil (Write One Bit)
06	Preset Single Register (Write One Word)
07	Read Exception Status (Read Error Code)
15	Force Multiple Coils (Write Multiple Bits)
16	Preset Multiple Registers (Write Words)
17	Report Slave ID

The DMC-30000 provides three levels of Modbus communication. The first level allows the user to create a raw packet and receive raw data. It uses the MBh command with a function code of –1. The format of the command is

MBh = -1,len,array[] where len is the number of bytes

array[] is the array with the data

The second level incorporates the Modbus structure. This is necessary for sending configuration and special commands to an I/O device. The formats vary depending on the function code that is called. For more information refer to the Command Reference.

The third level of Modbus communication uses standard Galil commands. Once the slave has been configured, the commands that may be used are @IN[], @AN[], SB, CB, OB, and AO. For example, AO 2020,8.2 would tell I/O number 2020 to output 8.2 volts.

If a specific slave address is not necessary, the I/O number to be used can be calculated with the following:

I/O Number = (HandleNum*1000) + ((Module-1)*4) + (BitNum-1)

Where HandleNum is the handle number from 1 (A) to 8 (8). Module is the position of the module in the rack from 1 to 16. BitNum is the I/O point in the module from 1 to 4.

Modbus Examples

Example #1

DMC-30000 connected as a Modbus master to a RIO-47120 via Modbus. The DMC-30000 will set or clear all 16 of the RIO's digital outputs

1. Begin by opening a connection to the RIO which in our example has IP address 192.168.1.120

```
IHB=192,168,1,120<502>2
```
(Issued to DMC-30000)

2. Dimension an array to store the commanded values. Set array element 0 equal to 170 and array element 1 equal to 85. (array element 1 configures digital outputs 15-8 and array element 0 configures digital outputs 7-0)

```
DM myarray[2]
myarray[0] = 170          (which is 10101010 in binary)
myarray[1] = 85           (which is 01010101in binary)
```

3. a) Send the appropriate MB command. Use function code 15. Start at output 0 and set/clear all 16 outputs based on the data in myarray[]

```
MBB=,15,0,16,myarray[]
```

3. b) Set the outputs using the SB command.

```
SB2001;SB2003;SB2005;SB2007;SB2008;SB2010;SB2012;SB2014;
```

Results:

Both steps 3a and 3b will result in outputs being activated as below. The only difference being that step 3a will set and clear all 16 bits where as step 3b will only set the specified bits and will have no affect on the others.

Bit Number	Status
0	0
1	1
2	0
3	1
4	0
5	1
6	0
7	1

Bit Number	Status
8	1
9	0
10	1
11	0
12	1
13	0
14	1
15	0

Example #2

DMC-30000 connected as a Modbus master to a 3rd party PLC. The DMC-30000 will read the value of analog inputs 3 and 4 on the PLC located at addresses 40006 and 40008 respectively. The PLC stores values as 32-bit floating point numbers which is common.

1. Begin by opening a connection to the PLC which has an IP address of 192.168.1.10 in our example

```
IHB=192,168,1,10<502>2
```

2. Dimension an array to store the results

```
DM myanalog[4]
```

3. Send the appropriate MB command. Use function code 4 (as specified per the PLC). Start at address 40006. Retrieve 4 modbus registers (2 modbus registers per 1 analog input, as specified by the PLC)

```
MBB=,4,40006,4,myanalog[]
```

Results:

Array elements 0 and 1 will make up the 32 bit floating point value for analog input 3 on the PLC and array elements 2 and 3 will combine for the value of analog input 4.

```
myanalog[0]=16412=0x401C
myanalog[1]=52429=0xCCCD
myanalog[2]=49347=0xC0C3
myanalog[3]=13107=0x3333
```

Analog input 3 = 0x401CCCCD = 2.45V

Analog input 4 = 0xC0C33333 = -6.1V

Example #3

DMC-30000 connected as a Modbus master to a hydraulic pump. The DMC-30000 will set the pump pressure by writing to an analog output on the pump located at Modbus address 30000 and consisting of 2 Modbus registers forming a 32 bit floating point value.

1. Begin by opening a connection to the pump which has an IP address of 192.168.1.100 in our example

```
IHB=192,168,1,100<502>2
```

2. Dimension and fill an array with values that will be written to the PLC

```
DM pump[2]
pump[0]=16531=0x4093
pump[1]=13107=0x3333
```

3. Send the appropriate MB command. Use function code 16. Start at address 30000 and write to 2 registers using the data in the array pump[]

```
MBB=,16,30000,2,pump[]
```

Results:

Analog output will be set to 0x40933333 which is 4.6V

To view an example procedure for communicating with an OPTO-22 rack, refer to List of Other Publications in the Appendices.

Data Record

The DMC-30000 can provide a binary block of status information with the use of the QR and DR commands. These commands along with the QZ command can be very useful for accessing complete controller status. The following is the byte map for the binary data. See the QR, QZ and DR command for specific command usage information.

NOTE: *UB = Unsigned Byte (1), UW = Unsigned Word (2), SW = Signed Word (2), SL = Signed Long Word (4), UL = Unsigned Long Word (4)*

ADDR	TYPE	ITEM
00	UB	1st Byte of Header
01	UB	2nd Byte of Header
02	UB	3rd Byte of Header
03	UB	4th Byte of Header
04-05	UW	sample number
06-07	UW	general input block 0 (inputs 1-16) [1]
08-09	UW	general output block 0 (outputs 1-16) [1]
10	UB	error code
11	UB	thread status – see bit field map below
12-13	UW	analog input 2
14-15	UW	analog output 1
16-17	UW	analog output 2
18-21	UL	amplifier status
22-25	UL	Segment Count for Contour Mode
26-27	UW	Buffer space remaining – Contour Mode
28-29	UW	segment count of coordinated move for S plane
30-31	UW	coordinated move status for S plane – see bit field map below
32-35	SL	distance traveled in coordinated move for S plane
36-37	UW	Buffer space remaining – S Plane
38-39	UW	A axis status – see bit field map below
40	UB	A axis switches – see bit field map below
41	UB	A axis stop code
42-45	SL	A axis reference position
46-49	SL	A axis motor position
50-53	SL	A axis position error
54-57	SL	A axis auxiliary position
58-61	SL	A axis velocity
62-65	SL	A axis torque
66-67	UW	analog input 1
68	UB	A Hall Input Status
69	UB	Reserved
70-73	SL	A User defined variable (ZA)

1 Not all I/O shown in the data record are available on the standard DMC-30000 controller. Contact a Galil Application Engineer for customization options.

Explanation Data Record Bit Fields

Header Information - Byte 0, 1 of Header:

BIT 15	BIT 14	BIT 13	BIT 12	BIT 11	BIT 10	BIT 9	BIT 8
1	N/A	N/A	N/A	N/A	I Block Present in Data Record	N/A	S Block Present in Data Record
BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
N/A	N/A	N/A	N/A	N/A	N/A	N/A	A Block Present in Data Record

Bytes 2, 3 of Header:

Bytes 2 and 3 make a word which represents the Number of bytes in the data record, including the header.

Byte 2 is the low byte and byte 3 is the high byte

NOTE: The header information of the data records is formatted in little endian (reversed network byte order).

Thread Status (1 Byte)

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
N/A	N/A	N/A	N/A	Thread 3 Running	Thread 2 Running	Thread 1 Running	Thread 0 Running

Coordinated Motion Status (2 Byte)

BIT 15	BIT 14	BIT 13	BIT 12	BIT 11	BIT 10	BIT 9	BIT 8
Move in Progress	N/A	N/A	N/A	N/A	N/A	N/A	N/A

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
N/A	N/A	Motion is slewing	Motion is stopping due to ST or Limit Switch	Motion is making final decel.	N/A	N/A	N/A

Axis Status (1 Word)

BIT 15	BIT 14	BIT 13	BIT 12	BIT 11	BIT 10	BIT 9	BIT 8
Move in Progress	Mode of Motion PA or PR	Mode of Motion PA only	(FE) Find Edge in Progress	Home (HM) in Progress	1st Phase of HM complete	2nd Phase of HM complete or FI command issued	Mode of Motion Coord. Motion

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
Negative Direction Move	Mode of Motion Contour	Motion is slewing	Motion is stopping due to ST of Limit Switch	Motion is making final decel.	Latch is armed	3rd Phase of HM in Progress	Motor Off

Axis Switches (1 Byte)

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
Latch Occurred	State of Latch Input	N/A	N/A	State of Forward Limit	State of Reverse Limit	State of Home Input	Stepper Mode

Notes Regarding Velocity and Torque Information

The velocity information that is returned in the data record is 64 times larger than the value returned when using the command TV (Tell Velocity). See command reference for more information about TV.

The Torque information is represented as a number in the range of +/-32767. Maximum negative torque is -32767. Maximum positive torque is 32767. Zero torque is 0.

QZ Command

The QZ command can be very useful when using the QR command, since it provides information about the controller and the data record. The QZ command returns the following 4 bytes of information.

BYTE #	INFORMATION
0	number of axes present ('1' for the DMC-30000)
1	number of bytes in general block of data record (18 for the DMC-30000)
2	number of bytes in coordinate plane block of data record (16 for the DMC-30000)
3	number of bytes the axis block of data record (36 for the DMC-30000)

GalilTools (Windows and Linux)

The DMC-30000 requires GalilTools version 1.6.0.0 or newer.

GalilTools is Galil's set of software tools for current Galil controllers. It is highly recommended for all first-time purchases of Galil controllers as it provides easy set-up, tuning and analysis. GalilTools replaces the WSDK Tuning software with an improved user-interface, real-time scopes and communications utilities.

The Galil Tools set contains the following tools: Scope, Editor, Terminal, Watch and Tuner, and a Communication Library for development with Galil Controllers.

The powerful Scope Tool is ideal for system analysis as it captures numerous types of data for each axis in real-time. Up to eight channels of data can be displayed at once, and additional real-time data can be viewed by changing the scope settings. This allows literally hundreds of parameters to be analyzed during a single data capture sequence. A rising or falling edge trigger feature is also included for precise synchronization of data.

The Program Editor Tool allows for easy writing of application programs and multiple editors to be open simultaneously.

The Terminal Tool provides a window for sending and receiving Galil commands and responses.

The Watch Tool displays controller parameters in a tabular format and includes units and scale factors for easy viewing.

The Tuning Tool helps select PID parameters for optimal servo performance.

The Communication Library provides function calls for communicating to Galil Controllers with C++ (Windows and Linux) and COM enabled languages such as VB C#, and Labview (Windows only).

GalilTools runs on Windows and Linux platforms as standard with other platforms available on request.

GalilTools-Lite is available at no charge and contains the Editor, Terminal, Watch and Communcition Library tools only.

The latest version of GalilTools can be downloaded from the Galil website at:

http://www.galilmc.com/products/software/galiltools.html

For information on using GalilTools see the help menu in GalilTools, or the GalilTools user manual.

http://www.galilmc.com/support/manuals/galiltools/index.html

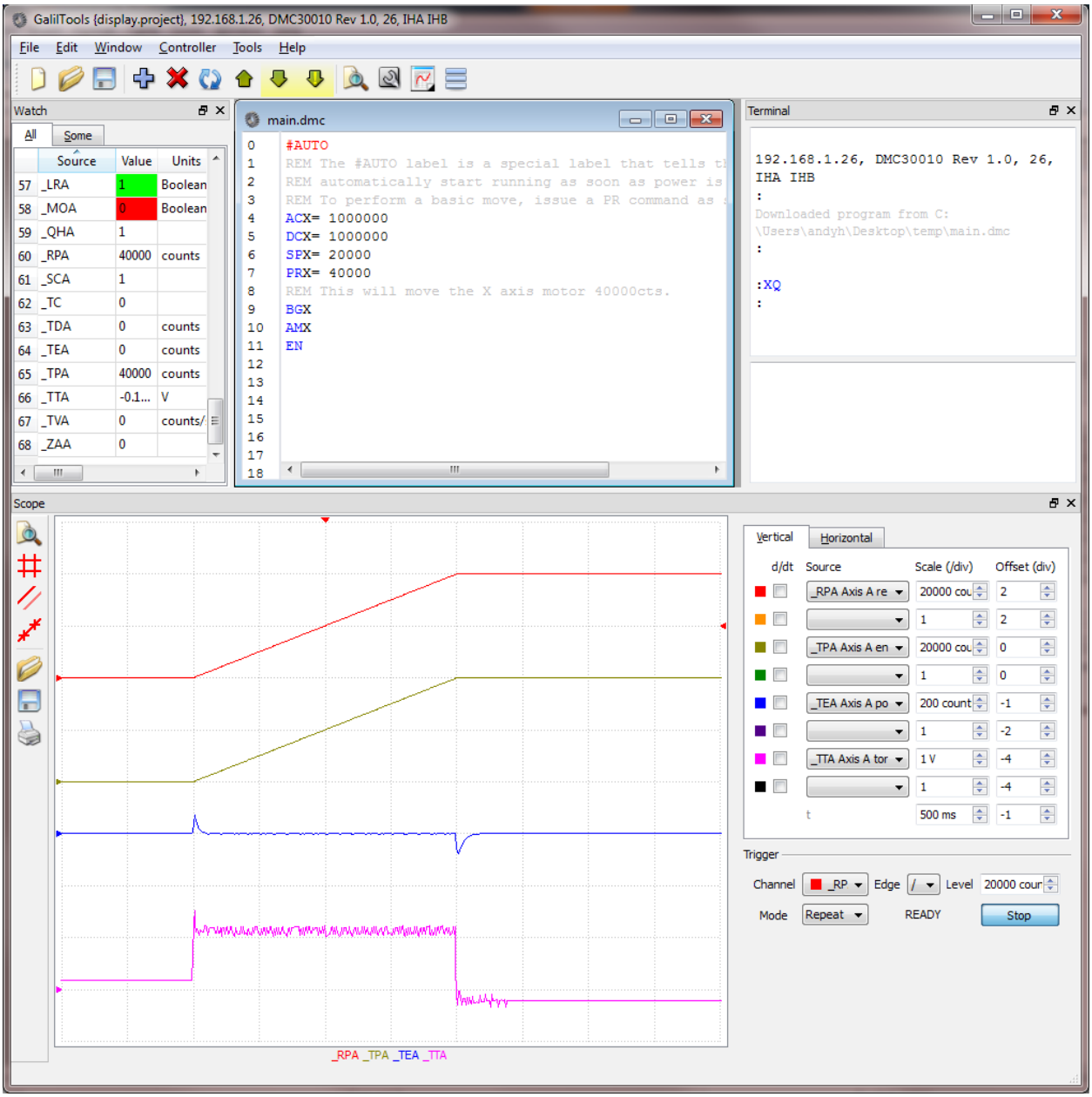


Figure 4.1: GalilTools Screen Capture

Creating Custom Software Interfaces

GalilTools provides a programming API so that users can develop their own custom software interfaces to a Galil controller. Information on this GalilTools Communication Library can be found in the GalilTools manual.

http://www.galilmc.com/support/manuals/galiltools/library.html

HelloGalil – Quick Start to PC programming

For programmers developing Windows applications that communicate with a Galil controller, the HelloGalil library of quick start projects immediately gets you communicating with the controller from the programming language of your choice. In the "Hello World" tradition, each project contains the bare minimum code to demonstrate communication to the controller and simply prints the controller's model and serial numbers to the screen (Figure 4.2):

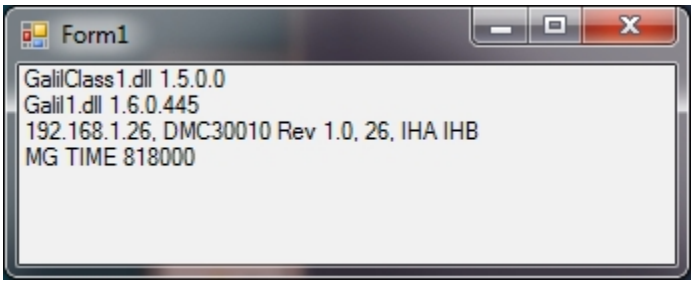


Figure 4.2: Sample program output

http://www.galilmc.com/support/hello_galil.html

GalilTools Communication Libraries

The GalilTools Communication Library (Galil class) provides methods for communication with a Galil motion controller over Ethernet, USB, RS-232 or PCI buses. It consists of a native C++ Library and a similar COM interface which extends compatibility to Windows programming languages (e.g. VB, C#, etc).

A Galil object (usually referred to in sample code as "g") represents a single connection to a Galil controller.

For Ethernet controllers, which support more than one connection, multiple objects may be used to communicate with the controller. An example of multiple objects is one Galil object containing a TCP handle to a DMC-30000 for commands and responses, and one Galil object containing a UDP handle for unsolicited messages from the controller. If recordsStart() is used to begin the automatic data record function, the library will open an additional UDP handle to the controller (transparent to the user).

The library is conceptually divided into six categories:

1. Connecting and Disconnecting - functions to establish and discontinue communication with a controller.
2. Basic Communication - The most heavily used functions for command-and-response and unsolicited messages.
3. Programs - Downloading and uploading embedded programs.
4. Arrays - Downloading and uploading array data.
5. Advanced - Lesser-used calls.
6. Data Record - Access to the data record in both synchronous and asynchronous modes.

C++ Library (Windows and Linux)

Both Full and Lite versions of GalilTools ship with a native C++ communication library. The Linux version (libGalil.so) is compatible with g++ and the Windows version (Galil1.dll) with Visual C++ 2008. Contact Galil if another version of the C++ library is required. See the getting started guide and the hello.cpp example in /lib.

COM (Windows)

To further extend the language compatibility on Windows, a COM (Component Object Model) class built on top of the C++ library is also provided with Windows releases. This COM wrapper can be used in any language and IDE supporting COM (Visual Studio 2005, 2008, etc). The COM wrapper includes all of the functionality of the base C++ class. See the getting started guide and the hello.* examples in \lib for more info.

For more information on the GalilTools Communications Library, see the online user manual.

http://www.galilmc.com/support/manuals/galiltools/library.html

Chapter 5 Command Basics

Introduction

The DMC-30000 provides over 100 commands for specifying motion and machine parameters. Commands are included to initiate action, interrogate status and configure the digital filter. These commands are sent in ASCII.

The DMC-30000 instruction set is BASIC-like and easy to use. Instructions consist of two uppercase letters that correspond phonetically with the appropriate function. For example, the instruction BG begins motion, and ST stops the motion.

Commands can be sent "live" over the communications port for immediate execution by the DMC-30000, or an entire group of commands can be downloaded into the DMC-30000 memory for execution at a later time. Combining commands into groups for later execution is referred to as Applications Programming and is discussed in the following chapter.

This section describes the DMC-30000 instruction set and syntax. A summary of commands as well as a complete listing of all DMC-30000 instructions is included in the *Command Reference*.

http://www.galilmc.com/support/manuals.php

Command Syntax - ASCII

DMC-30000 instructions are represented by two ASCII upper case characters followed by applicable arguments. A space may be inserted between the instruction and arguments. A semicolon or <return> is used to terminate the instruction for processing by the DMC-30000 command interpreter.

NOTE: If you are using a Galil terminal program, commands will not be processed until an <return> command is given. This allows the user to separate many commands on a single line and not begin execution until the user gives the <return> command.

IMPORTANT: All DMC-30000 commands are sent in upper case.

For example, the command

```
PR 4000 <return>        Position relative
```

Implicit Notation

PR is the two character instruction for position relative. 4000 is the argument which represents the required position value in counts. The <return> terminates the instruction. The space between PR and 4000 is optional.

To view the current values for each command, type the command followed by a ?.

```
PR 1000                         Specify a relative move
```

```
                                        of 1000
PR ?                                         Request relative move
                                        value
```

Explicit Notation

The DMC-30000 provides an alternative method for specifying data. Here data is specified individually using the single axis specifier A. An equals sign is used to assign data to that axis. For example:

```
PRA= 1000          Specify a position relative movement for the A axis of 1000
ACA= 200000        Specify acceleration as 200000
```

Controller Response to DATA

The DMC-30000 returns a : for valid commands and a ? for invalid commands.

For example, if the command BG is sent in lower case, the DMC-30000 will return a ?.

:bg	invalid command, lower case
?	DMC-30000 returns a ?

When the controller receives an invalid command the user can request the error code. The error code will specify the reason for the invalid command response. To request the error code type the command TC1. For example:

?TC1	Tell Code command
1 Unrecognized	Returned response

There are many reasons for receiving an invalid command response. The most common reasons are: unrecognized command (such as typographical entry or lower case), command given at improper time (such as during motion), or a command out of range (such as exceeding maximum speed). A complete listing of all codes is listed in the TC command in the Command Reference section.

Interrogating the Controller

Interrogation Commands

The DMC-30000 has a set of commands that directly interrogate the controller. When the command is entered, the requested data is returned in decimal format on the next line followed by a carriage return and line feed. The format of the returned data can be changed using the Position Format (PF), Variable Format (VF) and Leading Zeros (LZ) command. See Chapter 7 Application Programming and the Command Reference.

Summary of Interrogation Commands

RP	Report Command Position
RL	Report Latch
^R ^V	Firmware Revision Information
SC	Stop Code
TA	Tell Amplifier Error
TB	Tell Status
TC	Tell Error Code
TD	Tell Dual Encoder
TE	Tell Error
TI	Tell Input

TP	Tell Position
TR	Trace
TS	Tell Switches
TT	Tell Torque
TV	Tell Velocity

For example, the following example illustrates how to display the current position of the A axis:

```
TP A                    Tell position A
0                       Controllers Response
```

Interrogating Current Commanded Values.

Most commands can be interrogated by using a question mark (?) as the axis specifier.

```
PR ?                        Request PR setting
```

The controller can also be interrogated with operands.

Operands

Most DMC-30000 commands have corresponding operands that can be used for interrogation. Operands must be used inside of valid DMC expressions. For example, to display the value of an operand, the user could use the command:

MG ‘operand’ where ‘operand’ is a valid DMC operand

All of the command operands begin with the underscore character (_). For example, the value of the current position on the A axis can be assigned to the variable ‘V’ with the command:

V=_TPA

The Command Reference denotes all commands which have an equivalent operand as "Operand Usage". Also, see description of operands in Chapter 7 Application Programming.

Command Syntax – Binary (advanced)

Some commands have an equivalent binary value. Binary communication mode can be executed about 20% faster than ASCII commands. Binary format can only be used when commands are sent from the PC and cannot be embedded in an application program.

Binary Command Format

All binary commands have a 4 byte header and is followed by data fields. The 4 bytes are specified in hexadecimal format.

Header Format:

Byte 1 specifies the command number between 80 to FF. The complete binary command number table is listed below.

Byte 2 specifies the # of bytes in each field as 0,1,2,4 or 6 as follows:

00	No data fields (i.e. SH or BG)
01	One byte per field
02	One word (2 bytes per field)
04	One long word (4 bytes) per field
06	Galil real format (4 bytes integer and 2 bytes fraction)

Byte 3 specifies whether the command applies to a coordinated move as follows:

00	No coordinated motion movement
01	Coordinated motion movement

For example, the command STS designates motion to stop on a vector move, S coordinate system. The third byte for the equivalent binary command would be 01.

Byte 4 specifies the axis # or data field as follows

Bit 7 = H axis or 8th data field

Bit 6 = G axis or 7th data field

Bit 5 = F axis or 6th data field

Bit 4 = E axis or 5th data field

Bit 3 = D axis or 4th data field

Bit 2 = C axis or 3rd data field

Bit 1 = B axis or 2nd data field

Bit 0 = A axis or 1st data field

Data fields Format

Data fields must be consistent with the format byte and the axes byte. For example, the command PR 1000,, -500 would be :

```
A7 02 00 05 03 E8 FE 0C
```

where A7 is the command number for PR

02 specifies 2 bytes for each data field

00 S is not active for PR

05 specifies bit 0 is active for A axis and bit 2 is active for C axis ($2^0 + 2^2=5$)

03 E8 represents 1000

FE OC represents -500

Example

The command ST XS would be :

```
A1 00 01 01
```

where A1 is the command number for ST

00 specifies 0 data fields

01 specifies stop the coordinated axes S

01 specifies stop X (bit 0) $2^0 =1$

Binary command table

Command	No.	Command	No.	Command	No.
reserved	80	reserved	ab	reserved	d6
KP	81	reserved	ac	reserved	d7
KI	82	reserved	ad	RP	d8
KD	83	reserved	ae	TP	d9
DV	84	reserved	af	TE	da
AF	85	LM	b0	TD	db
KS	86	LI	b1	TV	dc
PL	87	VP	b2	RL	dd
ER	88	CR	b3	TT	de
IL	89	TN	b4	TS	df
TL	8a	LE, VE	b5	TI	e0
MT	8b	reserved	b6	SC	e1
CE	8c	VA	b7	reserved	e2
OE	8d	VD	b8	reserved	e3
FL	8e	VS	b9	reserved	e4
BL	8f	VR	ba	TM	e5
AC	90	reserved	bb	CN	e6
DC	91	reserved	bc	LZ	e7
SP	92	CM	bd	OP	e8
IT	93	CD	be	OB	e9
FA	94	DT	bf	SB	ea
FV	95	ET	c0	CB	eb
GR	96	EM	c1	II	ec
DP	97	EP	c2	EI	ed
DE	98	EG	c3	AL	ee

OF	99	EB	c4	reserved	ef
GM	9a	EQ	c5	reserved	f0
reserved	9b	EC	c6	reserved	f1
reserved	9c	reserved	c7	reserved	f2
reserved	9d	AM	c8	reserved	f3
reserved	9e	MC	c9	reserved	f4
reserved	9f	TW	ca	reserved	f5
BG	a0	MF	cb	reserved	f6
ST	a1	MR	cc	reserved	f7
AB	a2	AD	cd	reserved	f8
HM	a3	AP	ce	reserved	f9
FE	a4	AR	cf	reserved	fa
FI	a5	AS	d0	reserved	fb
PA	a6	AI	d1	reserved	fc
PR	a7	AT	d2	reserved	fd
JG	a8	WT	d3	reserved	fe
MO	a9	reserved	d4	reserved	ff
SH	aa	reserved	d5		

Chapter 6 Programming

Overview

The DMC-30000 provides several modes of motion, including independent positioning and jogging, coordinated motion, electronic cam motion, and electronic gearing. Each one of these modes is discussed in the following sections.

EXAMPLE APPLICATION	MODE OF MOTION	COMMANDS
Absolute or relative positioning where each axis is independent and follows prescribed velocity profile.	Independent Axis Positioning	PA,PR SP,AC,DC
Velocity control where no final endpoint is prescribed. Motion stops on Stop command.	Independent Jogging	JG AC,DC ST
Absolute positioning mode where absolute position targets may be sent to the controller while the axis is in motion.	Position Tracking	PA, PT SP AC, DC
Motion Path described as incremental position points versus time.	Error: Reference source not found	CM CD DT
Motion Path described as incremental position, velocity and delta time	PVT Mode	PV BT
Coordinated motion where path is described by linear segments.	Linear Interpolation Mode	LM LI, LE VS,VR VA,VD
2-D motion path consisting of arc segments and linear segments, such as engraving or quilting.	Vector Mode: Linear and Circular Interpolation Motion	VM VP CR VS,VR VA,VD VE
Electronic gearing where slave axes are scaled to master axis which can move in both directions.	Electronic Gearing	GA GD _GP GR GM (if gantry)

Master/slave where slave axes must follow a master such as conveyer speed.	Electronic Gearing and Ramped Gearing	GA GD _GP GR
Moving along arbitrary profiles or mathematically prescribed profiles such as sine or cosine trajectories.	Error: Reference source not found	CM CD DT
Teaching or Record and Play Back	Error: Reference source not found with Teach (Record and Play-Back)	CM CD DT RA RD RC
Backlash Correction	Dual Loop (Auxiliary Encoder)	DV
Following a trajectory based on a master encoder position	Electronic Cam	EA EM EP ET EB EG EQ
Smooth motion while operating in independent axis positioning	Motion Smoothing	IT
Smooth motion while operating in vector or linear interpolation positioning	Motion Smoothing	IT
Smooth motion while operating with stepper motors	Stepper Motion Smoothing	KS
Gantry - two axes are coupled by gantry	Electronic Gearing - Error: Reference source not found	GR GM

Independent Axis Positioning

In this mode, motion between the specified axes is independent, and each axis follows its own profile. The user specifies the desired absolute position (PA) or relative position (PR), slew speed (SP), acceleration ramp (AC), and deceleration ramp (DC), for each axis. On begin (BG), the DMC-30000 profiler generates the corresponding trapezoidal or triangular velocity profile and position trajectory. The controller determines a new command position along the trajectory every sample period until the specified profile is complete. Motion is complete when the last position command is sent by the DMC-30000 profiler. Note: The actual motor motion may not be complete when the profile has been completed, however, the next motion command may be specified.

The Begin (BG) command can be issued for all axes either simultaneously or independently. XYZ or W axis specifiers are required to select the axes for motion. When no axes are specified, this causes motion to begin on all axes.

The speed (SP) and the acceleration (AC) can be changed at any time during motion, however, the deceleration (DC) and position (PR or PA) cannot be changed until motion is complete. Remember, motion is complete when the profiler is finished, not when the actual motor is in position. The Stop command (ST) can be issued at any time to decelerate the motor to a stop before it reaches its final position.

An incremental position movement (IP) may be specified during motion as long as the additional move is in the same direction. Here, the user specifies the desired position increment, n. The new target is equal to the old target plus the increment, n. Upon receiving the IP command, a revised profile will be generated for motion towards the

new end position. The IP command does not require a begin. Note: If the motor is not moving, the IP command is equivalent to the PR and BG command combination.

Command Summary - Independent Axis

COMMAND	DESCRIPTION
PR x	Specifies relative distance
PA x	Specifies absolute position
SP x	Specifies slew speed
AC x	Specifies acceleration rate
DC x	Specifies deceleration rate
BG A	Starts motion
ST X	Stops motion before end of move
IP x	Changes position target
IT x	Time constant for independent motion smoothing
AM X	Trippoint for profiler complete
MC A	Trippoint for “in position”

The lower case specifiers (x) represent position values for each axis.

The DMC-30000 also allows use of explicit notation such as PRX=2000

Operand Summary - Independent Axis

OPERAND	DESCRIPTION
_ACA	Return acceleration rate
_DCA	Return deceleration rate
_SPA	Returns the speed
_PAA	Returns current destination if the axis is moving, otherwise returns the current commanded position if in a move.
_PRA	Returns current incremental distance

Example - Absolute Position Movement

```
PA 10000          Specify absolute position
AC 1000000        Acceleration
DC 1000000        Deceleration
SP 50000          Speed
BG X              Begin motion
```

Independent Jogging

The jog mode of motion is very flexible because speed, direction and acceleration can be changed during motion. The user specifies the jog speed (JG), acceleration (AC), and the deceleration (DC) rate. The direction of motion is specified by the sign of the JG parameters. When the begin command is given (BG), the motor accelerates up to speed and continues to jog at that speed until a new speed or stop (ST) command is issued. If the jog speed is changed during motion, the controller will make a accelerated (or decelerated) change to the new speed.

An instant change to the motor position can be made with the use of the IP command. Upon receiving this command, the controller commands the motor to a position which is equal to the specified increment plus the current position. This command is useful when trying to synchronize the position of two motors while they are moving.

Note that the controller operates as a closed-loop position controller while in the jog mode. The DMC-30000 converts the velocity profile into a position trajectory and a new position target is generated every sample period. This method of control results in precise speed regulation with phase lock accuracy.

Command Summary - Jogging

COMMAND	DESCRIPTION
AC x	Specifies acceleration rate
BG X	Begins motion
DC x	Specifies deceleration rate
IP x	Increments position instantly
IT x	Time constant for independent motion smoothing
JG +/-x	Specifies jog speed and direction
ST A	Stops motion

Parameters can be set with explicit notation such as JGA=2000.

Operand Summary - Independent Axis

OPERAND	DESCRIPTION
_ACA	Return acceleration rate
_DCA	Return deceleration rate
_SPA	Returns the jog speed
_TVA	Returns the actual velocity (averaged over 256 samples)

Example - Jog in X only

Jog motor at 50000 count/s.

```
#A
AC 20000          Specify acceleration of 20000 counts / sec
DC 20000          Specify deceleration of 20000 counts / sec
JG 50000          Specify jog speed
BG X              Begin motion
EN
```

Example - Joystick Jogging

The jog speed can also be changed using an analog input such as a joystick. Assume that for a 10 Volt input the speed must be 50000 counts/sec.

```
      #JOY              Label
      JG0               Set in Jog Mode
      BGX               Begin motion
      #B                Label for loop
      V1 =@AN[1]        Read analog input
      VEL=V1*5000       Compute speed
0/10
      JG VEL            Change JG speed
      JP #B             Loop
```

Position Tracking

The Galil controller may be placed in the position tracking mode to support changing the target of an absolute position move on the fly. New targets may be given in the same direction or the opposite direction of the current position target. The controller will then calculate a new trajectory based upon the new target and the acceleration, deceleration, and speed parameters that have been set. The motion profile in this mode is trapezoidal. There is not a set limit governing the rate at which the end point may be changed, however at the standard TM rate, the controller updates the position information at the rate of 1msec. The controller generates a profiled point every other sample, and linearly interpolates one sample between each profiled point. Some examples of applications that may use this mode are satellite tracking, missile tracking, random pattern polishing of mirrors or lenses, or any application that requires the ability to change the endpoint without completing the previous move.

The PA command is typically used to command the axis to a specific absolute position. For some applications such as tracking an object, the controller must proceed towards a target and have the ability to change the target during the move. In a tracking application, this could occur at any time during the move or at regularly scheduled intervals. For example if a robot was designed to follow a moving object at a specified distance and the path of the object wasn't known the robot would be required to constantly monitor the motion of the object that it was following. To remain within a specified distance it would also need to constantly update the position target it is moving towards. Galil motion controllers support this type of motion with the position tracking mode. This mode will allow scheduled or random updates to the current position target on the fly. Based on the new target the controller will either continue in the direction it is heading, change the direction it is moving, or decelerate to a stop.

The position tracking mode shouldn't be confused with the contour mode. The contour mode allows the user to generate custom profiles by updating the reference position at a specific time rate. In this mode, the position can be updated randomly or at a fixed time rate, but the velocity profile will always be trapezoidal with the parameters specified by AC, DC, and SP. Updating the position target at a specific rate will not allow the user to create a custom profile.

The following example will demonstrate the possible different motions that may be commanded by the controller in the position tracking mode. In this example, there is a host program that will generate the absolute position targets. The absolute target is determined based on the current information the host program has gathered on the object that it is tracking.

The controller must be placed in the position tracking mode to allow on the fly absolute position changes. This is performed with the PT command. To place the X axis in this mode the host would issue PT1 to the controller. The next step is to begin issuing PA command to the controller. The BG command isn't required in this mode, the SP, AC, and DC commands determine the shape of the trapezoidal velocity profile that the controller will use.

Example - Motion 1:

The host program determines that the first target for the controller to move to is located at 5000 encoder counts. The acceleration and deceleration should be set to 150,000 countts/sec2 and the velocity is set to 50,000 counts/sec. The command sequence to perform this is listed below.

```
#EX1
PT 1;'            Place the X axis in Position tracking mode
AC 150000;' Set the X axis acceleration to 150000 counts/sec2
DC 150000;' Set the X axis deceleration to 150000 counts/sec2
SP 50000;'  Set the X axis speed to 50000 counts/sec
PA 5000;'   Command the X axis to absolute position 5000 encoder counts
EN
```

The output from this code can be seen in Figure 6.1, a screen capture from the GalilTools scope.

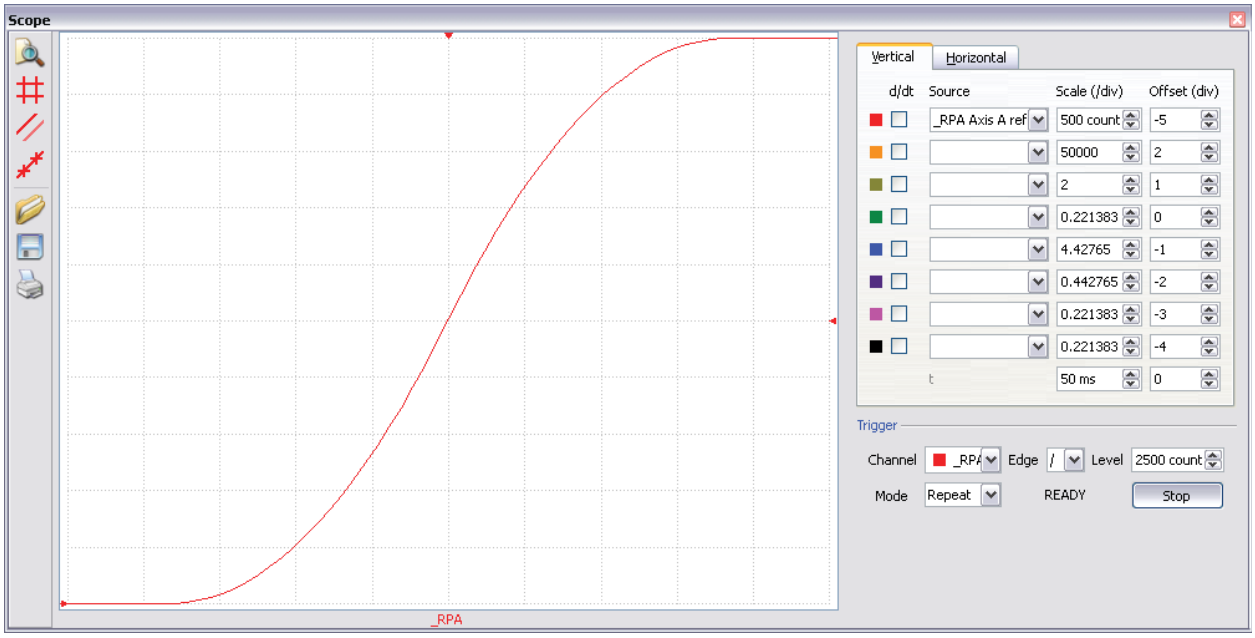


Figure 6.1: Position vs Time (msec) - Motion 1

Example - Motion 2:

The previous step showed the plot if the motion continued all the way to 5000, however partway through the motion, the object that was being tracked changed direction, so the host program determined that the actual target position should be 2000 counts at that time. Figure 6.1 shows what the position profile would look like if the move was allowed to complete to 5000 counts. The position was modified when the robot was at a position of 4200 counts(Figure 6.2). Note that the robot actually travels to a distance of almost 5000 counts before it turns around. This is a function of the deceleration rate set by the DC command. When a direction change is commanded, the controller decelerates at the rate specified by the DC command. The controller then ramps the velocity in up to the value set with SP in the opposite direction traveling to the new specified absolute position. In Figure 6.2 the velocity profile is triangular because the controller doesn't have sufficient time to reach the set speed of 50000 counts/sec before it is commanded to change direction.

The below code is used to simulate this scenario:

```
#EX2
PT 1;'      Place the X axis in Position tracking mode
AC 150000;'       Set the X axis acceleration to 150000 counts/sec2
DC 150000;'       Set the X axis deceleration to 150000 counts/sec2
SP 50000;'  Set the X axis speed to 50000 counts/sec
PA 5000;'   Command the X axis to abs position 5000 encoder counts
MF 4200
PA 2000;'   Change end point position to position 2000
EN
```

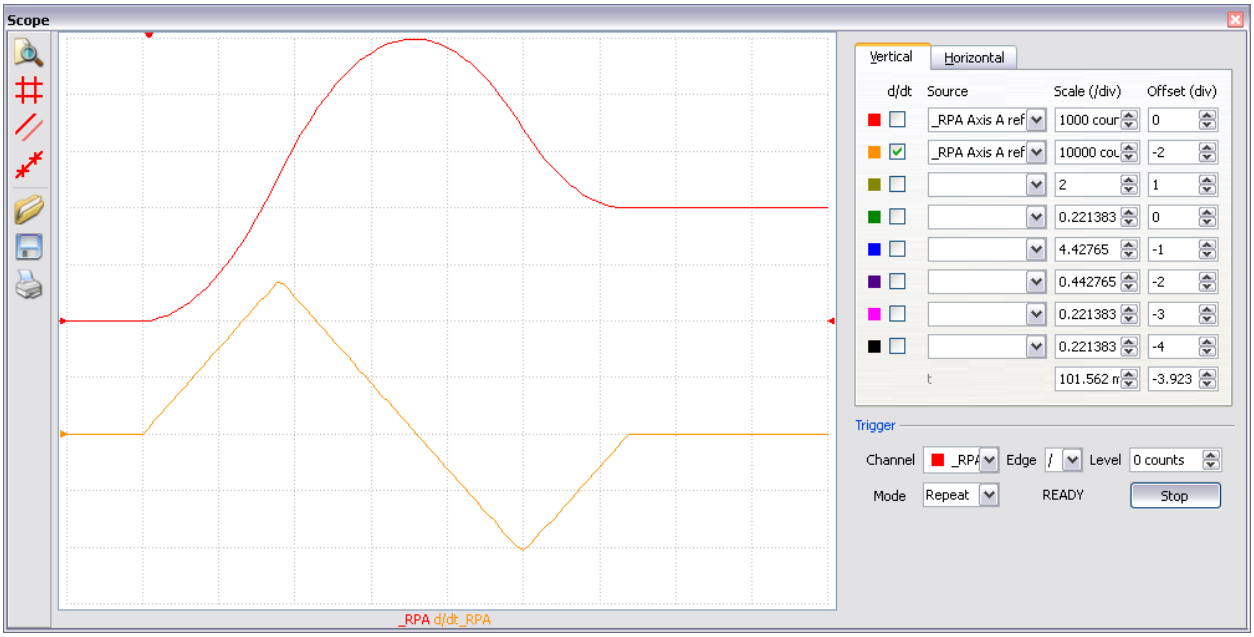


Figure 6.2: Position and Velocity vs Time(msec) for Motion 2

Example - Motion 3:

In this motion, the host program commands the controller to begin motion towards position 5000, changes the target to -2000, and then changes it again to 8000. Figure 6.3 shows the plot of position vs. time and velocity vs. time. Below is the code that is used to simulate this scenario:

```
#EX3
PT 1;' Place the X axis in Position tracking mode
AC 150000;' Set the X axis acceleration to 150000 counts/sec2
DC 150000;' Set the X axis deceleration to 150000 counts/sec2
SP 50000;' Set the X axis speed to 50000 counts/sec
PA 5000;' Command the X axis to abs position 5000 encoder counts
WT 300
PA -2000;' Change end point position to -2000
WT 200
PA 8000;' Change end point position to 8000
EN
```

Figure 6.4 demonstrates the use of motion smoothing (IT) on the velocity profile in this mode. The jerk in the system is also affected by the values set for AC and DC.

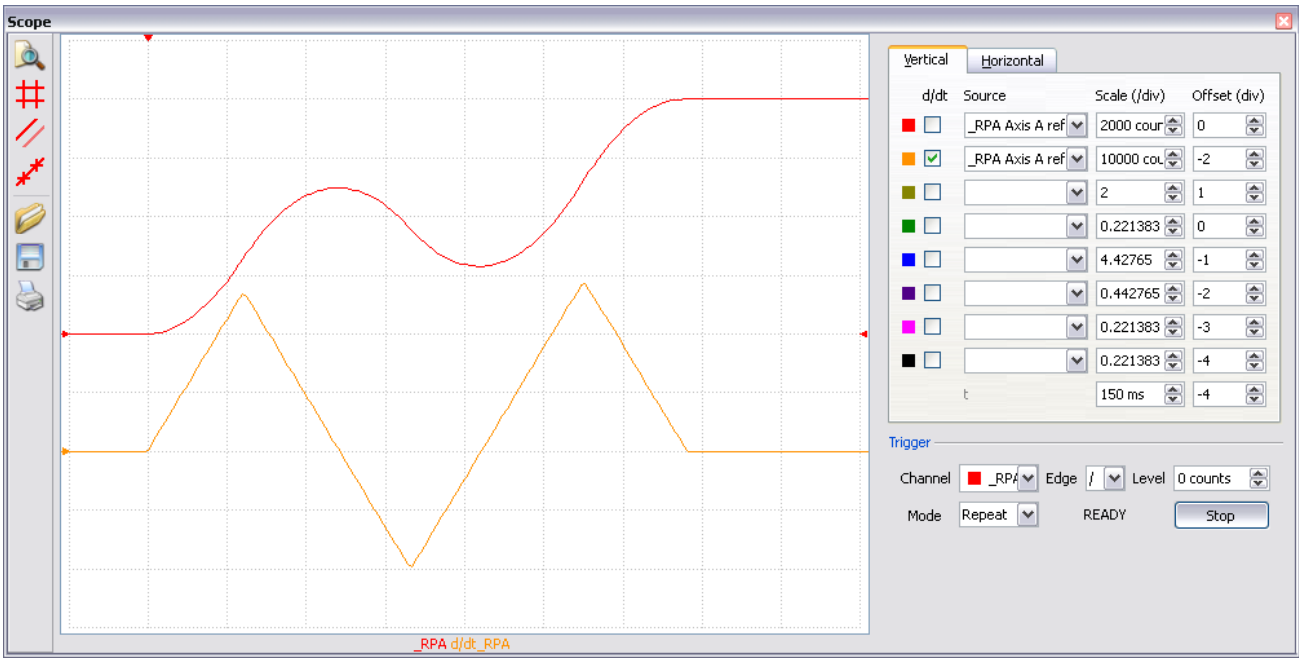


Figure 6.3: Position and Velocity vs Time (msec) for Motion 3

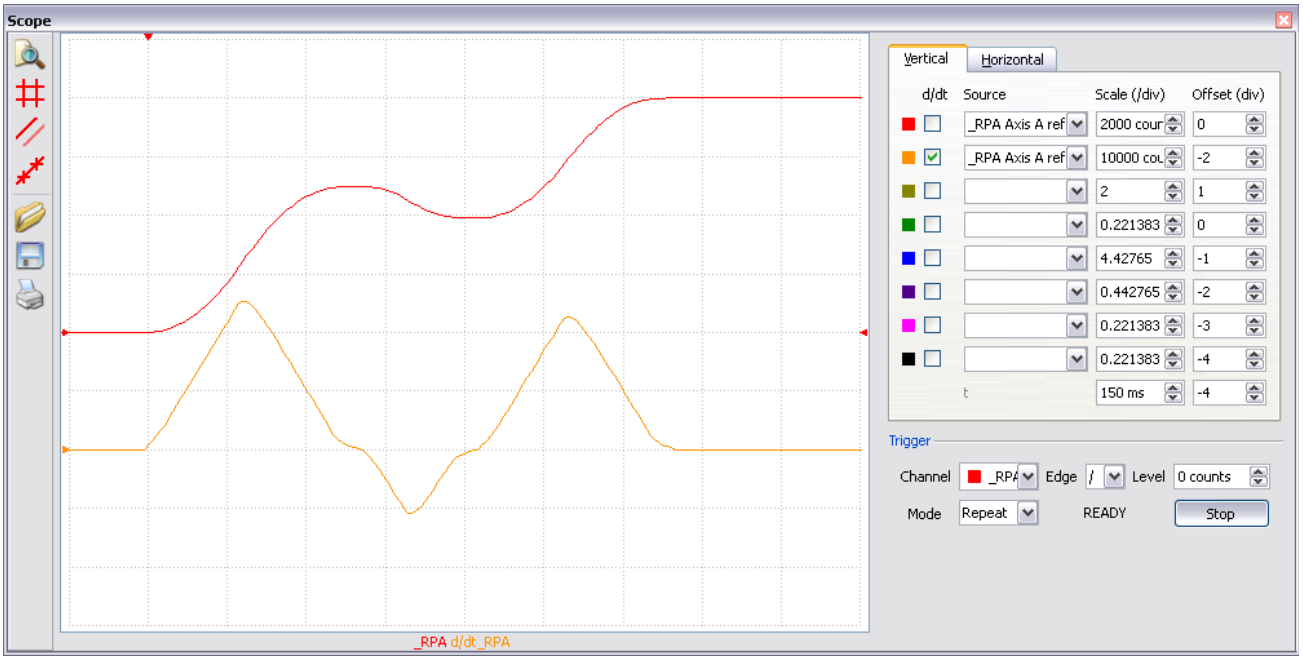


Figure 6.4: Position and Velocity vs Time (msec) for Motion 3 with IT 0.1

Note the controller treats the point where the velocity passes through zero as the end of one move, and the beginning of another move. IT is allowed, however it will introduce some time delay.

Trip Points

Most trip points are valid for use while in the position tracking mode. There are a few exceptions to this; the AM and MC commands may not be used while in this mode. It is recommended that AR, MF, MR, or AP be used, as they involve motion in a specified direction, or the passing of a specific absolute position.

Command Summary – Position Tracking Mode

COMMAND	DESCRIPTION
AC n	Acceleration settings
AP n	Trip point that holds up program execution until an absolute position has been reached
DC n	Deceleration settings
MF n	Trip point to hold up program execution until n number of counts have passed in the forward direction.
MR n	Trip point to hold up program execution until n number of counts have passed in the reverse direction.
PT n	Command used to enter and exit the Trajectory Modification Mode
PA n	Command Used to specify the absolute position target
SP n	Speed settings

Linear Interpolation Mode

The DMC-30000 provides a linear interpolation mode that allows the buffering of relative moves for a single axis. In linear interpolation mode the motion path is described in terms of incremental distances for each axis. An unlimited number of incremental segments may be given in a continuous move sequence, making the linear interpolation mode ideal for following a piece-wise linear path. There is no limit to the total move length.

The LM ("LM A") command selects the Linear Interpolation mode.

Specifying Linear Segments

The command LI x specifies the incremental move distance. This means motion is prescribed with respect to the current axis position. Up to 31 incremental move segments may be given prior to the Begin Sequence (BGS) command. Once motion has begun, additional LI segments may be sent to the controller.

The clear sequence (CS) command can be used to remove LI segments stored in the buffer prior to the start of the motion. To stop the motion, use the instructions STS or AB. The command, ST, causes a decelerated stop. The command, AB, causes an instantaneous stop and aborts the program, and the command AB1 aborts the motion only.

The Linear End (LE) command must be used to specify the end of a linear move sequence. This command tells the controller to decelerate to a stop following the last LI command. If an LE command is not given, an Abort AB1 must be used to abort the motion sequence.

It is the responsibility of the user to keep enough LI segments in the DMC-30000 sequence buffer to ensure continuous motion. If the controller receives no additional LI segments and no LE command, the controller will stop motion instantly at the last vector. There will be no controlled deceleration. LM? or _LM returns the available spaces for LI segments that can be sent to the buffer. 31 returned means the buffer is empty and 31 LI segments can be sent. A zero means the buffer is full and no additional segments can be sent. As long as the buffer is not full, additional LI segments can be sent.

The instruction _CS returns the segment counter. As the segments are processed, _CS increases, starting at zero. This function allows the host computer to determine which segment is being processed.

Additional Commands

The commands VS n, VA n, and VD n are used to specify the vector speed, acceleration and deceleration.

An Example of Linear Interpolation Motion:

```
#LMOVE              label
DP 0                Define position of 0
LM X                Enable LM mode
LI 5000             Specify first linear segment
LI -10000           Specify second linear segment
LE                  End linear segments
VS 4000             Specify vector speed
BG S                Begin motion sequence
EN                  Program end
```

Specifying Vector Speed for Each Segment

The instruction VS has an immediate effect and, therefore, must be given at the required time. In some applications, it is necessary to attach various speeds to different motion segments. This can be done by two functions: < n and > m

For example: LI x < n >m

The first command, < n, is equivalent to commanding VS n at the start of the given segment and will cause an acceleration toward the new commanded speeds, subjects to the other constraints.

The second function, > m, requires the vector speed to reach the value m at the end of the segment. Note that the function > m may start the deceleration within the given segment or during previous segments, as needed to meet the final speed requirement, under the given values of VA and VD.

Note, however, that the controller works with one > m command at a time. As a consequence, one function may be masked by another. For example, if the function >100000 is followed by >5000, and the distance for deceleration is not sufficient, the second condition will not be met. The controller will attempt to lower the speed to 5000, but will reach that at a different point.

As an example, consider the following program.

```
#ALT                Label for alternative program
DP 0                Define Position of 0
LM XY               Enable LM mode
LI 4000<4000>1000   Specify first linear segment with a vector speed of
                    4000 and end speed 1000
LI 1000<4000>1000   Specify second linear segment with a vector speed of
                    4000 and end speed 1000
LI 5000<4000>1000   Specify third linear segment with a vector speed of
                    4000 and end speed 1000
LE                  End linear segments
BG S                Begin motion sequence
EN                  Program end
```

Changing Feed Rate:

The command VR n allows the feed rate, VS, to be scaled between 0 and 10 with a resolution of .0001. This command takes effect immediately and causes VS to be scaled. VR also applies when the vector speed is specified with the '<' operator. This is a useful feature for feed rate override. VR does not ratio the accelerations. For example, VR .5 results in the specification VS 2000 to be divided in half.

Command Summary - Linear Interpolation

COMMAND	DESCRIPTION
LM A	Enable linear interpolation
LM ? or _LMS	Returns number of available spaces for linear segments in DMC-30000 sequence buffer. Zero means buffer full. 31 means buffer empty.
LI x<n>m	Specify incremental distances relative to current position, and assign vector speed n and m.
VS n	Specify vector speed
VA n	Specify vector acceleration
VD n	Specify vector deceleration
VR n	Specify the vector speed ratio
BG S	Begin Linear Sequence
CS	Clear sequence
LE	Linear End- Required at end of LI command sequence
LE ?	Returns the length of the vector (resets after 2147483647)
AM S	Trippoint for After Sequence complete
AV n	Trippoint for After Relative Vector distance, n

Operand Summary - Linear Interpolation

OPERAND	DESCRIPTION
_AV	Return distance traveled
_CS	Segment counter - returns number of the segment in the sequence, starting at zero.
_LE	Returns length of vector (resets after 2147483647)
_LM	Returns number of available spaces for linear segments in DMC-30000 sequence buffer. Zero means buffer full. 31 means buffer empty.
_VPA	Return the absolute coordinate of the last data point along the trajectory.

To illustrate the ability to interrogate the motion status, consider the first motion segment of our example, #LMOVE, where the X axis moves toward the point X=5000. Suppose that when X=3000, the controller is interrogated using the command 'MG _AV'. The returned value will be 3000. The value of _CS and _VPA will be zero.

Vector Mode: Linear and Circular Interpolation Motion

he DMC-30000 provides a vector mode that allows the buffering of absolute moves (from the starting position) for a single axis.

The coordinated motion mode is similar to the linear interpolation mode, but the linear segments are specified as absolute positions from the starting position of the A axis.

The command VM AN where A is the A axis, and N is the imaginary axis.

Specifying Vector Segments

The motion segments are described by two commands; VP for linear segments and CR for circular segments. Once a set of linear segments and/or circular segments have been specified, the sequence is ended with the command VE. This defines a sequence of commands for coordinated motion. Immediately prior to the execution of the first coordinated movement, the controller defines the current position to be zero for all movements in a sequence. Note: This 'local' definition of zero does not affect the absolute coordinate system or subsequent coordinated motion sequences.

The command, VP x,y specifies the coordinates of the end points of the vector movement with respect to the starting point. The command, CR r,q,d define a circular arc with a radius r, starting angle of q, and a traversed angle d. The notation for q is that zero corresponds to the positive horizontal direction, and for both q and d, the counter-clockwise (CCW) rotation is positive. The CR command is useful for producing a sine wave as the move output as a single axis of a circle is a sinusoidal profile.

Up to 31 segments of CR or VP may be specified in a single sequence and must be ended with the command VE. The motion can be initiated with a Begin Sequence (BGS) command. Once motion starts, additional segments may be added.

The Clear Sequence (CS) command can be used to remove previous VP and CR commands which were stored in the buffer prior to the start of the motion. To stop the motion, use the instructions STS or AB1. ST stops motion at the specified deceleration. AB1 aborts the motion instantaneously.

The Vector End (VE) command must be used to specify the end of the coordinated motion. This command requires the controller to decelerate to a stop following the last motion requirement. If a VE command is not given, an Abort (AB1) must be used to abort the coordinated motion sequence.

It is the responsibility of the user to keep enough motion segments in the DMC-30000 sequence buffer to ensure continuous motion. If the controller receives no additional motion segments and no VE command, the controller will stop motion instantly at the last vector. There will be no controlled deceleration. LM? or _LM returns the available spaces for motion segments that can be sent to the buffer. 31 returned means the buffer is empty and 31 segments can be sent. A zero means the buffer is full and no additional segments can be sent. As long as the buffer is not full, additional segments can be sent at PC bus speeds.

The operand _CS can be used to determine the value of the segment counter.

Additional commands

The commands VS n, VA n and VD n are used for specifying the vector speed, acceleration, and deceleration.

Specifying Vector Speed for Each Segment:

The vector speed may be specified by the immediate command VS. It can also be attached to a motion segment with the instructions

VP x,y < n >m

CR r,θ,δ < n >m

The first command, <n, is equivalent to commanding VS n at the start of the given segment and will cause an acceleration toward the new commanded speeds, subjects to the other constraints.

The second function, > m, requires the vector speed to reach the value m at the end of the segment. Note that the function > m may start the deceleration within the given segment or during previous segments, as needed to meet the final speed requirement, under the given values of VA and VD.

Note, however, that the controller works with one > m command at a time. As a consequence, one function may be masked by another. For example, if the function >100000 is followed by >5000, and the distance for deceleration is not sufficient, the second condition will not be met. The controller will attempt to lower the speed to 5000, but will reach that at a different point.

Changing Feed Rate:

The command VR n allows the feed rate, VS, to be scaled between 0 and 10 with a resolution of .0001. This command takes effect immediately and causes VS scaled. VR also applies when the vector speed is specified with the '<' operator. This is a useful feature for feed rate override. VR does not ratio the accelerations. For example, VR 0.5 results in the specification VS 2000 to be divided by two.

Trippoints:

The AV n command is the After Vector trippoint, which waits for the vector relative distance of n to occur before executing the next command in a program.

Command Summary - Coordinated Motion Sequence

COMMAND	DESCRIPTION.
VM AN	Enable Vector Mode
VP m,n	Specify the Vector segment
CR r,Θ, ±ΔΘ	Specifies arc segment where r is the radius, Θ is the starting angle and ΔΘ is the travel angle. Positive direction is CCW.
VS s	Specify vector speed or feed rate of sequence.
VA s	Specify vector acceleration along the sequence.
VD s	Specify vector deceleration along the sequence.
VR s	Specify vector speed ratio
BG S	Begin motion sequence
CS S	Clear sequence
AV s	Trippoint for After Relative Vector distance.
AMST	Holds execution of next command until Motion Sequence is complete.
LM?	Return number of available spaces for linear and circular segments in DMC-30000 sequence buffer. Zero means buffer is full. 31 means buffer is empty.

Operand Summary - Coordinated Motion Sequence

OPERAND	DESCRIPTION
_VPA	The absolute coordinate of the axis at the last intersection along the sequence.
_AV	Distance traveled.
_LM	Number of available spaces for linear and circular segments in DMC-30000 sequence buffer. Zero means buffer is full. 31 means buffer is empty.

_CS	Segment counter - Number of the segment in the sequence, starting at zero.
_VE	Vector length of coordinated move sequence.

When AV is used as an operand, _AV returns the distance traveled along the sequence.

The operand _VPX can be used to return the coordinates of the last point specified along the path.

Example (Sine Wave Output):

The CR command can be used to command sinusoidal motion to the axis. The below code and scope output shown in Figure 6.5 show an example of how this can be achieved. The frequency and amplitude of the output can be modified by changing the radius in the CR command and by changing the vector speed.

```
REM frequency output (Hz) = (VS*(pi/2))/(r*10)
REM ex VS 12000 and r=1590
REM frequency(Hz)=(12000*1.57)/(1590*10)=1.18Hz
#SinWv
'vector speed
VS12000
'1/2 amplitude of sine wave
r=1590
VM AN
CR r,-90,90
CR r,0,720
CR r,0,720
CR r,0,720
VE
BGS
'Continue to create sine wave
#LOOP
CR r,0,720;CR r,0,720
#wt;JP#wt,_LM<30
JP#LOOP
```

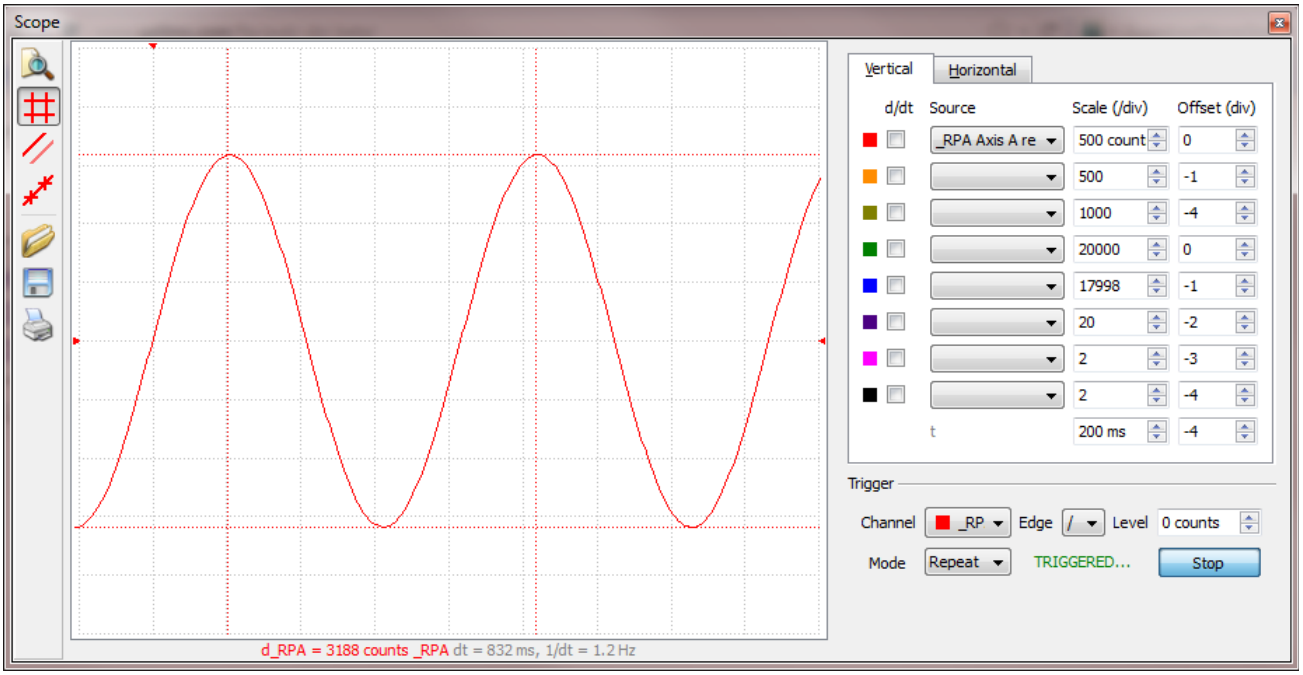


Figure 6.5: Sine Output with Vector Mode

Electronic Gearing

This mode allows up the axis to be electronically geared to the Auxiliary encoder or the imaginary axis. The master may rotate in both directions and the axis will follow at the specified gear ratio. The gear ratio may be changed during motion.

The GA command specifies the master axes and the GR command specifies the gear ratio for the slave where the ratio may be a number between +/-127.9999 with a fractional resolution of .0001. There are two modes: standard gearing and gantry mode. The gantry mode (enabled with the command GM) allows the gearing to stay enabled even if a limit is hit or an ST command is issued. GR 0 turns off gearing in both modes.

Electronic gearing allows the geared motor to perform a second independent move in addition to the gearing. For example, when a geared motor follows a master at a ratio of 1:1, it may be advanced an additional distance with PR, or JG, commands, or VP, or LI.

Ramped Gearing

In some applications, especially when the master is traveling at high speeds, it is desirable to have the gear ratio ramp gradually to minimize large changes in velocity on the slave when the gearing is engaged. For example if the master is already traveling at 500,000 counts/sec and the slave will be geared at a ratio of 1:1 when the gearing is engaged, the slave will instantly develop following error, and command maximum current to the motor. This can be a large shock to the system. For many applications it is acceptable to slowly ramp the engagement of gearing over a greater time frame. Galil allows the user to specify an interval of the master axis over which the gearing will be engaged. For example, the same master X axis in this case travels at 500,000 counts/sec, and the gear ratio is 1:1, but the gearing is slowly engaged over 30,000 counts of the master axis, greatly diminishing the initial shock to the slave axis. Figure 6.6 below shows the velocity vs. time profile for instantaneous gearing. Figure 6.7 shows the velocity vs. time profile for the gradual gearing engagement.

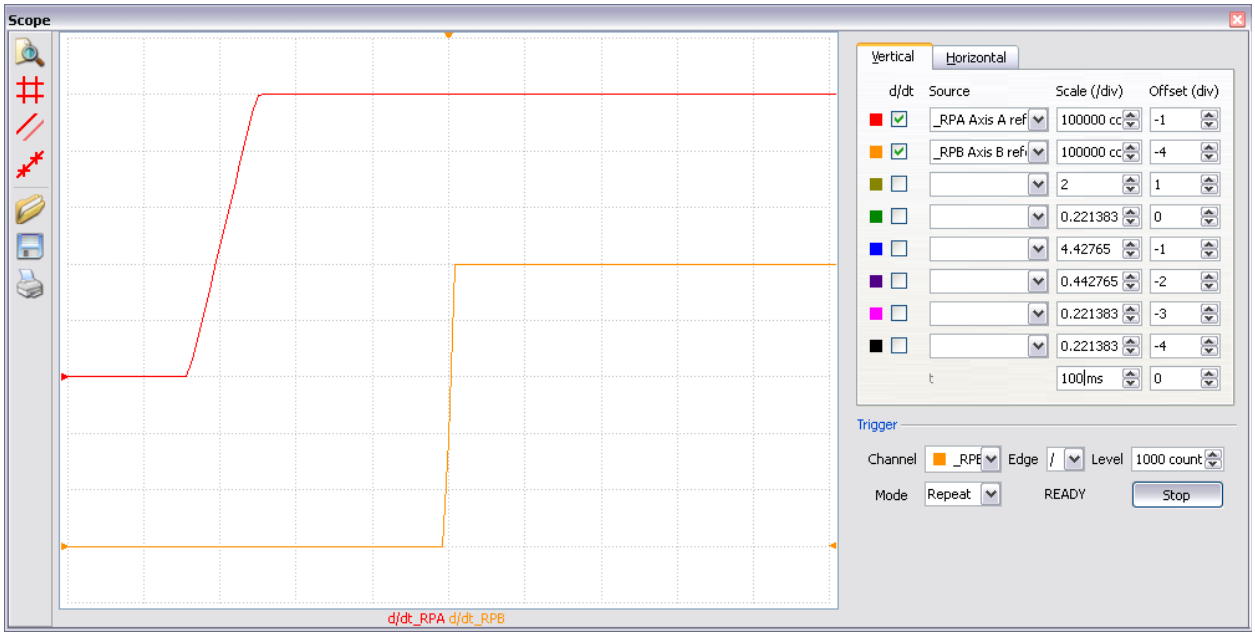


Figure 6.6: Velocity counts/sec vs. Time (msec) Instantaneous Gearing Engagement

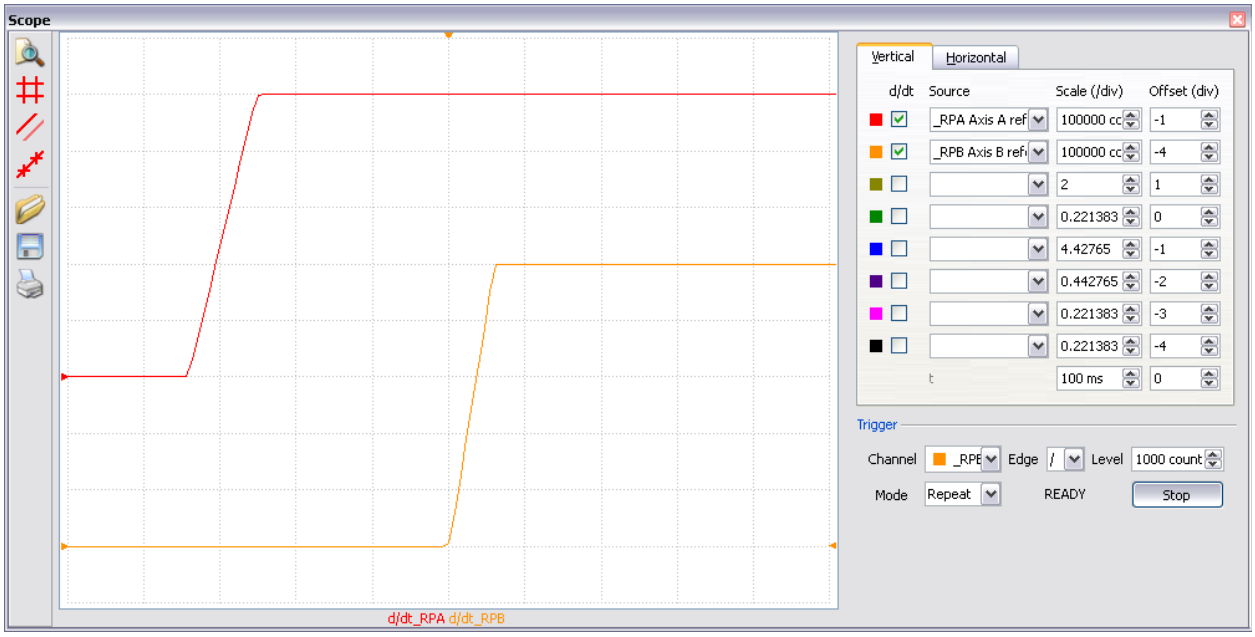


Figure 6.7: Velocity (counts/sec) vs. Time (msec) Ramped Gearing

The slave axis for each figure is shown on the bottom portion of the figure; the master axis is shown on the top portion. The shock to the slave axis will be significantly less in Figure 6.7 than in Figure 6.6. The ramped gearing does have one consequence. There isn't a true synchronization of the two axes, until the gearing ramp is complete. The slave will lag behind the true ratio during the ramp period. If exact position synchronization is required from the point gearing is initiated, then the position must be commanded in addition to the gearing. The controller keeps track of this position phase lag with the _GP operand. The following example will demonstrate how the command is used.

Command Summary - Electronic Gearing

COMMAND	DESCRIPTION
GA n	Specifies master axes for gearing where: n = DA, S or N
GD a	Sets the distance the master will travel for the gearing change to take full effect.
_GPA	This operand keeps track of the difference between the theoretical distance traveled if gearing changes took effect immediately, and the distance traveled since gearing changes take effect over a specified interval.
GR a	Sets gear ratio. 0 disables electronic gearing for specified axis.
GM a	a = 1 sets gantry mode, 0 disables gantry mode
MR x	Trippoint for reverse motion past specified value.
MF x	Trippoint for forward motion past specified value.

Example - Simple Master Slave

Master axis is the imaginary axis and moves 10000 counts, A axis will move 50000 counts.

```
GA N              Specify master axes as the N axis
GR 5              Set gear ratio of 5x
PRN=10000         Specify N position
BGN               Begin motion
```

Electronic Cam

The electronic cam is a motion control mode which enables the periodic synchronization the motor. The master axis encoder can be the auxiliary encoder input or the virtual axis.

The electronic cam is a more general type of electronic gearing which allows a table-based relationship between the axes. It allows synchronizing all the controller axes.

To illustrate the procedure of setting the cam mode, consider the cam relationship shown in Figure 6.8.

Step 1. Selecting the master axis

The first step in the electronic cam mode is to select the master axis. This is done with the instruction

EAp where p = DA or N

p is the selected master axis

For the given example, since the master is the aux encoder input, we specify EA DA

Step 2. Specify the master cycle and the change in the slave axis (or axes).

In the electronic cam mode, the position of the master is always expressed modulo one cycle. In this example, the position of the master axis is always expressed in the range between 0 and 6000. Similarly, the slave position is also redefined such that it starts at zero and ends at 1500. At the end of a cycle when the master is 6000 and the slave is 1500, the positions of both x and y are redefined as zero. The MM command specifies the master modulus, and the EM command specifies the slave modulus.

The cycle of the master is limited to 8,388,607 whereas the slave change per cycle is limited to 2,147,483,647. If the change is a negative number, the absolute value is specified. For the given example, the cycle of the master is 6000 counts and the change in the slave is 1500. Therefore, we use the instructions:

```
MM 6000
EM 1500
```

Step 3. Specify the master interval and starting point.

Next we need to construct the ECAM table. The table is specified at uniform intervals of master positions. Up to 256 intervals are allowed. The size of the master interval and the starting point are specified by the instruction:

```
EP n0,n1
```

where n0 is the interval width in counts, and n1 is the phase shift.

For the given example, we can specify the table by specifying the position at the master points of 0, 2000, 4000 and 6000. We can specify that by

```
EP 2000,0
```

Step 4. Specify the slave positions.

Next, we specify the slave positions with the instruction

```
ET[n]=x
```

where n indicates the order of the point.

The value, n, starts at zero and may go up to 256. The parameter x will indicate the corresponding slave position. For this example, the table may be specified by

```
ET[0]=0
ET[1]=3000
ET[2]=2250
```

```
ET[3]=1500
```

This specifies the ECAM table.

Step 5. Enable the ECAM

To enable the ECAM mode, use the command

```
EB n
```

where n=1 enables ECAM mode and n=0 disables ECAM mode.

Step 6. Engage the slave motion

To engage the slave motion, use the instruction

```
EG x
```

where x is the master positions at which the corresponding slave must be engaged.

If the value of any parameter is outside the range of one cycle, the cam engages immediately. When the cam is engaged, the slave position is redefined, modulo one cycle.

Step 7. Disengage the slave motion

To disengage the cam, use the command

```
EQ x
```

where x is the master positions at which the corresponding slave axis is disengaged.

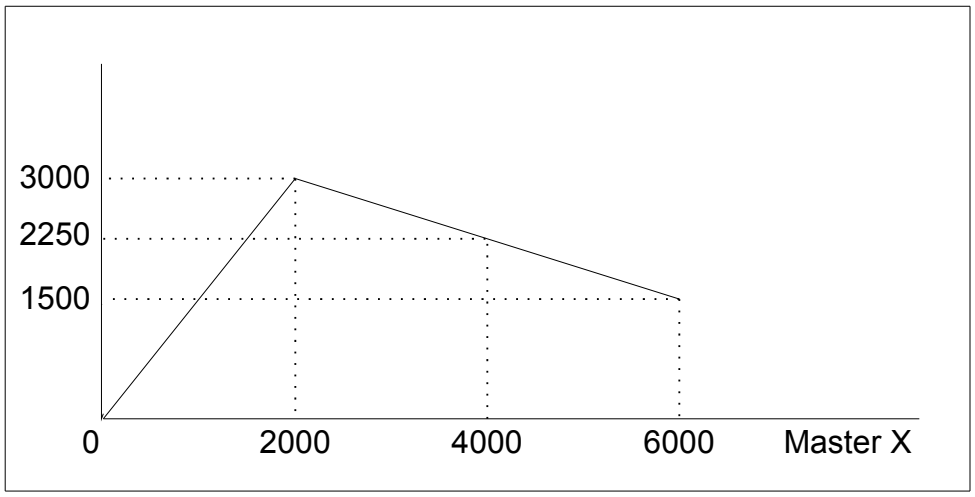


Figure 6.8: Electronic Cam Example

This disengages the slave axis at a specified master position. If the parameter is outside the master cycle, the stopping is instantaneous.

ECAM - Example

To illustrate the complete process, consider the cam relationship described by

the equation:

$X = 0.5 * N + 100 \sin (0.18*N)$

where N (virtual axis) is the master, with a cycle of 2000 counts.

The cam table can be constructed manually, point by point, or automatically by a program. The following program includes the set-up.

The instruction EA N defines virtual axis as the master axis. The cycle of the master is 2000. Over that cycle, the slave varies by 1000. This leads to the instructions MMN= 2000 and EMA= 1000.

The following routine computes the table points. As the phase equals 0.18X and X varies in increments of 20, the phase varies by increments of 3.6°. The program then computes the values of Y according to the equation and assigns the values to the table with the instruction ET[i] = x.

INSTRUCTION	INTERPRETATION
#SETUP	Label
EAN	Select X as master
EMA= 1000	Slave Modulus
MMN=2000	
EP 20,0	Master position increments
i = 0	Index
#LOOP	Loop to construct table from equation
p = i*3.6	Note 3.6 = 0.18 * 20
s = @SIN[p]*100	Define sine position
x = i*10+s	Define slave position
ET [i] = x	Define table
i = i+1	
JP #LOOP, i<=100	Repeat the process
EN	

PVT Mode

The DMC-30000 controllers now supports a mode of motion referred to as "PVT." This mode allows arbitrary motion profiles to be defined by position, velocity and time. This motion is designed for systems where the load must traverse a series of coordinates with no discontinuities in velocity. By specifying the target position, velocity and time to achieve those parameters the user has control over the velocity profile. Taking advantage of the built in buffering the user can create virtually any profile including those with infinite path lengths.

Specifying PVT Segments

PVT segments are commanded using the PV command. The PV command includes the target distance to be moved and target velocity to be obtained over the specified timeframe. Positions are entered as relative moves, similar to the standard PR command, in units of encoder counts and velocity is entered in counts/second. The controller will interpolate the motion profile between subsequent PV commands using a 3rd order polynomial equation. During a PV segment, jerk is held constant, and accelerations, velocities, and positions will be calculated every other sample.

Motion will not begin until a BT command is issued, much like the standard BG command. This means that the user can fill the PVT buffer prior to motion beginning. PVT mode has a 127 segment buffer. This buffer is a FIFO and the available space can be queried with the operand _PVA. As the buffer empties the user can add more PVT segments by issuing new PV commands.

Exiting PVT Mode

To exit PVT mode the user must send the segment command PVA=0,0,0. This will exit the mode once the segment is reached in the buffer. To avoid an abrupt stop the user should slow the motion to a zero velocity prior to executing this command. The controller will instantly command a zero velocity once a PVA=0,0,0 is executed. In addition, a ST command will also exit PVT mode. Motion will come to a controlled stop using the DC value for deceleration. The same controlled stop will occur if a limit switch is activated in the direction of motion. As a result, the controller will be switched to a jog mode of motion.

Error Conditions and Stop Codes

If the buffer is allowed to empty while in PVT mode then the profiling will be aborted and the motor will come to a controlled stop on that axis with a deceleration specified by the DC command. Also, PVT mode will be exited and the stop code will be set to 32. During normal operation of PVT mode the stop code will be 30. If PVT mode is exited normally (PVA=0,0,0), then the stop code will be set to 31.

Additional PVT Information

It is the users' responsibility to enter PVT data that the system's mechanics and power system can respond to in a reasonable manner. Because this mode of motion is not constrained by the AC, DC or SP values, if a large velocity or position is entered with a short period to achieve it, the acceleration can be very high, beyond the capabilities of the system, resulting in excessive position error. The position and velocity at the end of the segment are guaranteed to be accurate but it is important to remember that the required path to obtain the position and velocity in the specified time may be different based on the PVT values. Mismatched values for PVT can result in different interpolated profiles than expected but the final velocity and position will be accurate.

The "t" value is entered in samples, which will depend on the TM setting. With the default TM of 1000, one sample is 976us. This means that a "t" value of 1024 will yield one second of motion. The velocity value, "v" will always be in units of counts per second, regardless of the TM setting.

Command Summary – PVT

COMMAND	DESCRIPTION
PVA = p,v,t	Specifies the segment for an incremental PVT segment of 'p' counts, an end speed of 'v' counts/sec in a total time of 't' samples.
_PVA	Contains the number of PV segments available in the PV buffer.
BT	Begin PVT mode
_BTA	Contains the number PV segments that have executed

PVT Examples

Parabolic Velocity Profile

In this example we will assume that the user wants to start from zero velocity, accelerate to a maximum velocity of 1000 counts/second in 1 second and then back down to 0 counts/second within an additional second. The velocity profile would be described by the following equation and shown in Figure 6.9.

$$v(t) = -1000(t-1)^2 + 1000$$

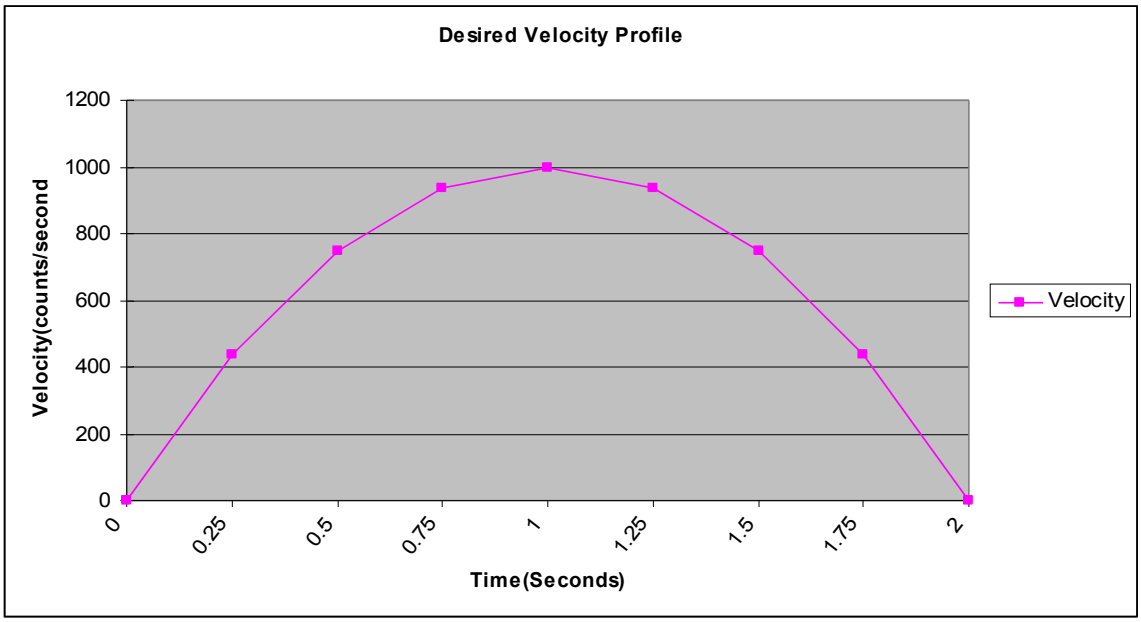


Figure 6.9: Parabolic Velocity Profile

To accomplish this we need to calculate the desired velocities and change in positions. In this example we will assume a delta time of ¼ of a second, which is 256 samples (1024 samples = 1 second with the default TM of 1000).

Velocity(counts/second)	Position(counts)
$v(t) = -1000(t-1)^2 + 1000$	$p(t) = \int (-1000(t-1)^2 + 1000)dt$

$v(.25) = 437.5$	$p(0 \text{ to } .25) = 57$
$v(.5) = 750$	$p(.25 \text{ to } .5) = 151$
$v(.75) = 937.5$	$p(.5 \text{ to } .75) = 214$
$v(1) = 1000$	$p(.75 \text{ to } 1) = 245$
$v(1.25) = 937.5$	$p(1 \text{ to } 1.25) = 245$
$v(1.5) = 750$	$p(1.25 \text{ to } 1.5) = 214$
$v(1.75) = 437.5$	$p(1.5 \text{ to } 1.75) = 151$
$v(2) = 0$	$p(1.75 \text{ to } 2) = 57$

The DMC program is shown below and the results can be seen in Figure 6.10.

```
INSTRUCTION          INTERPRETATION
     #PVT                 Label
     PVX =                Incremental move of 57 counts in 256
57,437,256           samples with a final velocity of 437 counts/sec
     PVX =                Incremental move of 151 counts in 256
151,750,256          samples with a final velocity of 750 counts/sec
     PVX =                Incremental move of 214 counts in 256
214,937,256          samples with a final velocity of 937 counts/sec
     PVX =                Incremental move of 245 counts in 256
245,1000,256         samples with a final velocity of 1000 counts/sec
     PVX =                Incremental move of 245 counts in 256
245,937,256          samples with a final velocity of 937 counts/sec
     PVX =                Incremental move of 214 counts in 256
214,750,256          samples with a final velocity of 750 counts/sec
     PVX =                Incremental move of 151 counts in 256
151,437,256          samples with a final velocity of 437 counts/sec
     PVX =                Incremental move of 57 counts in 256
57,0,256             samples with a final velocity of 0 counts/sec
     PVX =                Termination of PVT buffer
0,0,0
     BTX                  Begin PVT
     EN
```

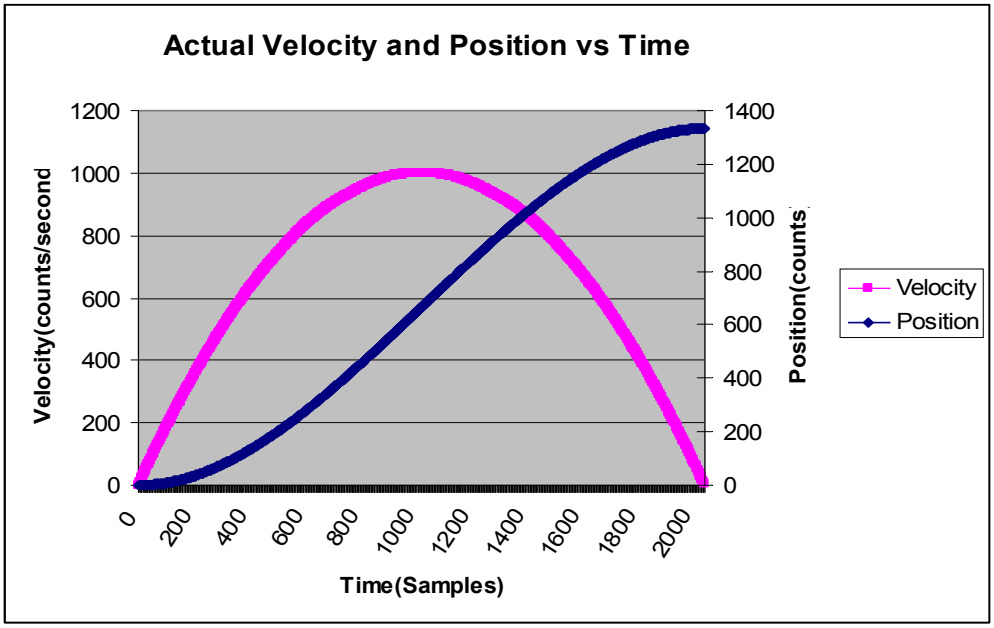


Figure 6.10: Actual Velocity and Position vs Time of Parabolic Velocity Profile

Contour Mode

The DMC-30000 also provides a contouring mode. This mode allows any arbitrary position curve to be prescribed. This is ideal for following computer generated paths such as parabolic, spherical or user-defined profiles. The path is not limited to straight line and arc segments and the path length may be infinite.

Specifying Contour Segments

The Contour Mode is specified with the command, CM.

A contour is described by position increments which are described with the command, CD x over a time interval, DT n. The parameter, n, specifies the time interval. The time interval is defined as 2^n sample period (1 ms for TM1000), where n is a number between 1 and 8. The controller performs linear interpolation between the specified increments, where one point is generated for each sample. If the time interval changes for each segment, use

CD x=n

where n is the new DT value.

Consider, for example, the trajectory shown in Figure 6.11. The position X may be described by the points:

Point 1	X=0 at T=0ms
Point 2	X=48 at T=4ms
Point 3	X=288 at T=12ms
Point 4	X=336 at T=28ms

The same trajectory may be represented by the increments

Increment 1	DX=48	Time=4	DT=2
Increment 2	DX=240	Time=8	DT=3
Increment 3	DX=48	Time=16	DT=4

When the controller receives the command to generate a trajectory along these points, it interpolates linearly between the points. The resulting interpolated points include the position 12 at 1 msec, position 24 at 2 msec, etc.

The programmed commands to specify the above example are:

```
#A
CMX                 Specifies X axis for contour mode
CD 48=2             Specifies first position increment and time
                  interval, 2^2 ms
CD 240=3            Specifies second position increment and time
                  interval, 2^3 ms
CD 48=4             Specifies the third position increment and
                  time interval, 2^4 ms
CD 0=0              End Contour buffer
#Wait;JP#Wait,      Wait until path is done
_CM<>511
EN
```

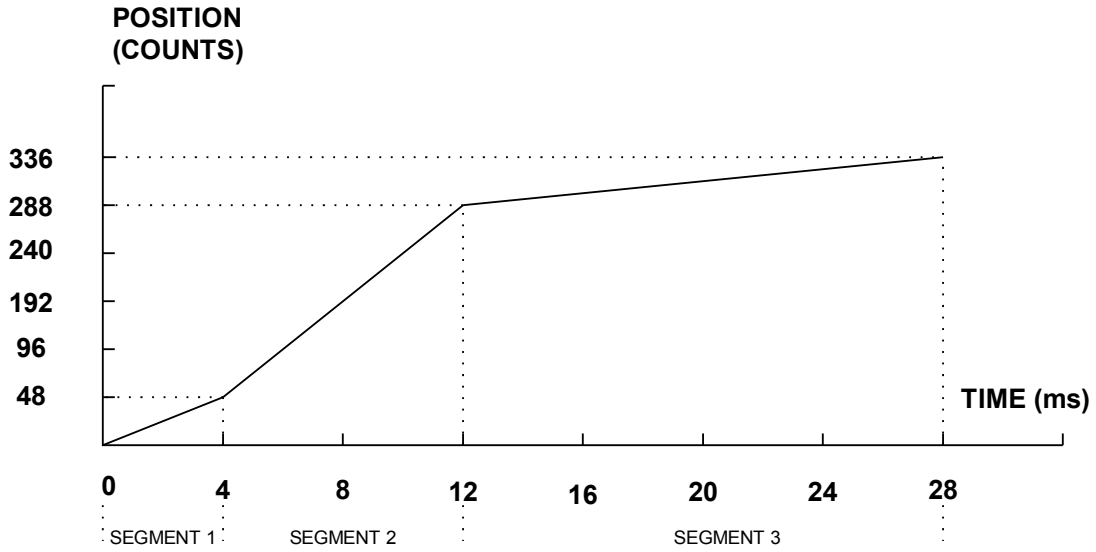


Figure 6.11: The Required Trajectory

Additional Commands

_CM gives the amount of space available in the contour buffer (511 maximum). Zero parameters for DT followed by zero parameters for CD will exit the contour mode.

If no new data record is found and the controller is still in the contour mode, the controller waits for new data. No new motion commands are generated while waiting. If bad data is received, the controller responds with a ?.

Specifying a -1 for the DT or as the time interval in the CD command will pause the contour buffer.

Issuing the CM command will clear the contour buffer.

Command Summary - Contour Mode

COMMAND	DESCRIPTION

CM A	Specifies contour mode.
CD x	Specifies position increment over time interval. Range is +/-32,000. CD 0=0 ends the contour buffer. This is much like the LE or VE commands.
DT n	Specifies time interval 2^n sample periods (1 ms for TM1000) for position increment, where n is an integer between 1 and 8. Zero ends contour mode. If n does not change, it does not need to be specified with each CD.
_CM	Amount of space left in contour buffer (511 maximum)

General Velocity Profiles

The Contour Mode is ideal for generating any arbitrary velocity profiles. The velocity profile can be specified as a mathematical function or as a collection of points.

The design includes two parts: Generating an array with data points and running the program.

Generating an Array - An Example

Consider the velocity and position profiles shown in Figure 6.12. The objective is to rotate a motor a distance of 6000 counts in 120 ms. The velocity profile is sinusoidal to reduce the jerk and the system vibration. If we describe the position displacement in terms of A counts in B milliseconds, we can describe the motion in the following manner:

$$\omega = \frac{A}{B}\left(1 - \cos(2\pi / B)\right)$$

$$X = \frac{AT}{B} - \frac{A}{2\pi}\sin(2\pi / B)$$

Note: ω is the angular velocity; X is the position; and T is the variable, time, in milliseconds.

In the given example, A=6000 and B=120, the position and velocity profiles are:

$X = 50T - (6000/2\pi) \sin (2\pi T/120)$

Note that the velocity, ω, in count/ms, is

$\omega = 50 [1 - \cos 2\pi T/120]$

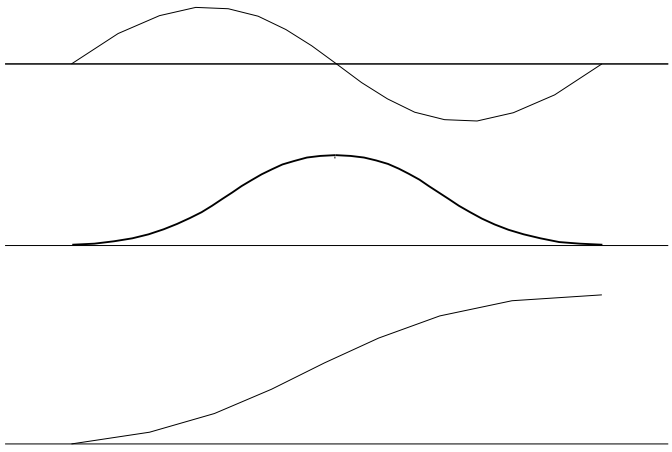


Figure 6.12: Velocity Profile with Sinusoidal Acceleration

The DMC-30000 can compute trigonometric functions. However, the argument must be expressed in degrees. Using our example, the equation for X is written as:

$X = 50T - 955 \sin 3T$

A complete program to generate the contour movement in this example is given below. To generate an array, we compute the position value at intervals of 8 ms. This is stored at the array POS. Then, the difference between the positions is computed and is stored in the array DIF. Finally the motors are run in the contour mode.

Contour Mode Example

```
INSTRUCTION          INTERPRETATION
     #POINTS              Program defines X points
     DM POS[16]           Allocate memory
     DM DIF[15]
     C=0                  Set initial conditions, C is index
     T=0                  T is time in ms
     #A
     V1=50*T
     V2=3*T               Argument in degrees
     V3=-                 Compute position
955*@SIN[V2]+V1
     V4=@INT[V3]          Integer value of V3
     POS[C]=V4            Store in array POS
     T=T+8
     C=C+1
     JP #A,C<16
     #B                   Program to find position differences
     C=0
     #C
     D=C+1
     DIF[C]=POS[D         Compute the difference and store
]-POS[C]
     C=C+1
     JP #C,C<15

     #RUN                 Program to run motor
     CMX                  Contour Mode
     DT3                  8 millisecond intervals
     C=0
     #E
     CD DIF[C]            Contour Distance is in DIF
     C=C+1
     JP #E,C<15
     CD 0=0               End contour buffer
     #Wait;JP#Wai         Wait until path is done
t,_CM<>511
     EN                   End the program
```

Teach (Record and Play-Back)

Several applications require teaching the machine a motion trajectory. Teaching can be accomplished using the DMC-30000 automatic array capture feature to capture position data. The captured data may then be played back in the contour mode. The following array commands are used:

DM C[n]	Dimension array
RA C[]	Specify array for automatic record (up to 4 for DMC-30000)
RD _TPX	Specify data for capturing (such as _TPX or _TPZ)
RC n,m	Specify capture time interval where n is 2^n sample periods (1 ms for TM1000), m is number of records to be captured
RC? or _RC	Returns a 1 if recording

Record and Playback Example:

```
#RECORD             Begin Program
DM XPOS[501]        Dimension array with 501 elements
RA XPOS[]           Specify automatic record
RD _TPX             Specify X position to be captured
MOX                 Turn X motor off
RC2                 Begin recording; 4 msec interval (at
                  TM1000)
#A;JP#A,_RC=        Continue until done recording
1
#COMPUTE            Compute DX
DM DX[500]          Dimension Array for DX
C=0                 Initialize counter
#L                  Label
D=C+1
DELTA=XPOS[D        Compute the difference
]-XPOS[C]
DX[C]=DELTA         Store difference in array
C=C+1               Increment index
JP #L,C<500         Repeat until done
#PLAYBCK            Begin Playback
CMX                 Specify contour mode
DT2                 Specify time increment
I=0                 Initialize array counter
#B                  Loop counter
CD DX[I];           Specify contour data I=I+1 Increment array
I=I+1               counter
JP #B,I<500         Loop until done
CD 0=0              End countour buffer
```

```
      #Wait;JP#Wai        Wait until path is done
t,_CM<>511
      EN                  End program
```

For additional information about automatic array capture, see Chapter 7 Application Programming.

Virtual Axis

The DMC-30000 controller has an additional virtual axis designated as the N axis. This axis has no encoder and no DAC. However, it can be commanded by the commands:

```
AC, DC, JG, SP, PR, PA, BG, IT, GA, VM, VP, CR, ST, DP, RP
```

The main use of the virtual axis is to serve as a virtual master in ECAM mode, and to perform an unnecessary part of a vector mode. These applications are illustrated by the following examples.

ECAM Master Example

Suppose that the motion of the XY axes is constrained along a path that can be described by an electronic cam table. Further assume that the ecam master is not an external encoder but has to be a controlled variable.

This can be achieved by defining the N axis as the master with the command EAN and setting the modulo of the master with a command such as EMN= 4000. Next, the table is constructed. To move the constrained axes, simply command the N axis in the jog mode or with the PR and PA commands.

For example,

```
PAN = 2000
BGN
```

will cause the XY axes to move to the corresponding points on the motion cycle.

Sinusoidal Motion Example

The x axis must perform a sinusoidal motion of 10 cycles with an amplitude of 1000 counts and a frequency of 20 Hz.

This can be performed by commanding the X and N axes to perform circular motion. Note that the value of VS must be

$$VS=2\pi * R * F$$

where R is the radius, or amplitude and F is the frequency in Hz.

Set VA and VD to maximum values for the fastest acceleration.

```
INSTRUCTION          INTERPRETATION
      VMXN                Select Axes
      VA 68000000         Maximum Acceleration
      VD 68000000         Maximum Deceleration
      VS 125664           VS for 20 Hz
      CR 1000, -90,       Ten Cycles
3600
      VE
      BGS
```

Stepper Motor Operation

When configured for stepper motor operation, several commands are interpreted differently than from servo mode. The following describes operation with stepper motors.

Specifying Stepper Motor Operation

Stepper motor operation is specified by the command MT. The argument for MT is as follows:

2 specifies a stepper motor with active low step output pulses
-2 specifies a stepper motor with active high step output pulses
2.5 specifies a stepper motor with active low step output pulses and reversed direction
-2.5 specifies a stepper motor with active high step output pulse and reversed direction

Stepper Motor Smoothing

The command, KS, provides stepper motor smoothing. The effect of the smoothing can be thought of as a simple Resistor-Capacitor (single pole) filter. The filter occurs after the motion profiler and has the effect of smoothing out the spacing of pulses for a more smooth operation of the stepper motor. Use of KS is most applicable when operating in full step or half step operation. KS will cause the step pulses to be delayed in accordance with the time constant specified.

When operating with stepper motors, you will always have some amount of stepper motor smoothing, KS. Since this filtering effect occurs after the profiler, the profiler may be ready for additional moves before all of the step pulses have gone through the filter. It is important to consider this effect since steps may be lost if the controller is commanded to generate an additional move before the previous move has been completed. See the discussion below, Monitoring Generated Pulses vs. Commanded Pulses.

The general motion smoothing command, IT, can also be used. The purpose of the command, IT, is to smooth out the motion profile and decrease 'jerk' due to acceleration.

Monitoring Generated Pulses vs. Commanded Pulses

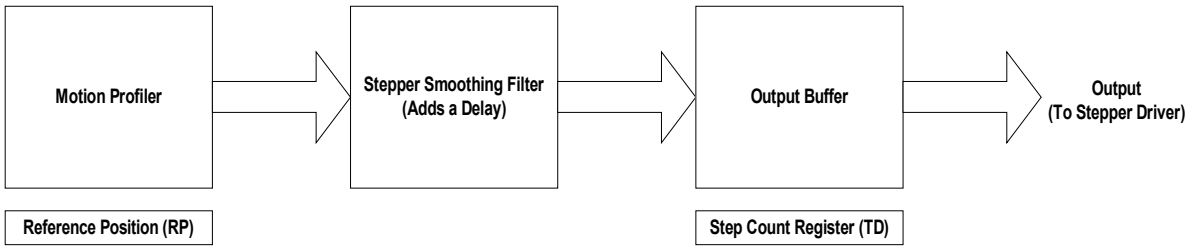
For proper controller operation, it is necessary to make sure that the controller has completed generating all step pulses before making additional moves. This is most particularly important if you are moving back and forth. For example, when operating with servo motors, the trippoint AM (After Motion) is used to determine when the motion profiler is complete and is prepared to execute a new motion command. However when operating in stepper mode, the controller may still be generating step pulses when the motion profiler is complete. This is caused by the stepper motor smoothing filter, KS. To understand this, consider the steps the controller executes to generate step pulses:

First, the controller generates a motion profile in accordance with the motion commands.

Second, the profiler generates pulses as prescribed by the motion profile. The pulses that are generated by the motion profiler can be monitored by the command, RP (Reference Position). RP gives the absolute value of the position as determined by the motion profiler. The command, DP, can be used to set the value of the reference position. For example, DP 0, defines the reference position of the X axis to be zero.

Third, the output of the motion profiler is filtered by the stepper smoothing filter. This filter adds a delay in the output of the stepper motor pulses. The amount of delay depends on the parameter which is specified by the command, KS. As mentioned earlier, there will always be some amount of stepper motor smoothing.

Fourth, the output of the stepper smoothing filter is buffered and is available for input to the stepper motor driver. The pulses which are generated by the smoothing filter can be monitored by the command, TD (Tell Dual). TD gives the absolute value of the position as determined by actual output of the buffer. The command, DP sets the value of the step count register as well as the value of the reference position. For example, DP 0, defines the reference position of the X axis to be zero.



Motion Complete Trippoint

When used in stepper mode, the MC command will hold up execution of the proceeding commands until the controller has generated the same number of steps out of the step count register as specified in the commanded position. The MC trippoint (Motion Complete) is generally more useful than AM trippoint (After Motion) since the step pulses can be delayed from the commanded position due to stepper motor smoothing.

Using an Encoder with Stepper Motors

An encoder may be used on a stepper motor to check the actual motor position with the commanded position. If an encoder is used, it must be connected to the main encoder input. Note: The auxiliary encoder is not available while operating with stepper motors. The position of the encoder can be interrogated by using the command, TP. The position value can be defined by using the command, DE.

Note: Closed loop operation with a stepper motor is not possible.

Command Summary - Stepper Motor Operation

COMMAND	DESCRIPTION
DE	Define Encoder Position (When using an encoder)
DP	Define Reference Position and Step Count Register
IT	Motion Profile Smoothing - Independent Time Constant
KS	Stepper Motor Smoothing
MT	Motor Type (2,-2,2.5 or -2.5 for stepper motors)
RP	Report Commanded Position
TD	Report number of step pulses generated by controller
TP	Tell Position of Encoder

Operand Summary - Stepper Motor Operation

OPERAND	DESCRIPTION
_DEA	Contains the value of the step count register
_DPA	Contains the value of the main encoder
_ITA	Contains the value of the Independent Time constant
_KSA	Contains the value of the Stepper Motor Smoothing constant
_MTA	Contains the motor type value
_RPA	Contains the commanded position generated by the profiler
_TDA	Contains the value of the step count register
_TPA	Contains the value of the main encoder

Stepper Position Maintenance Mode (SPM)

The Galil controller can be set into the Stepper Position Maintenance (SPM) mode to handle the event of stepper motor position error. The mode looks at position feedback from the main encoder and compares it to the commanded step pulses. The position information is used to determine if there is any significant difference between the commanded and the actual motor positions. If such error is detected, it is updated into a command value for operator use. In addition, the SPM mode can be used as a method to correct for friction at the end of a microstepping move. This capability provides closed-loop control at the application program level. SPM mode can be used with Galil and non-Galil step drives.

SPM mode is configured, executed, and managed with seven commands. This mode also utilizes the #POSERR automatic subroutine allowing for automatic user-defined handling of an error event.

Internal Controller Commands (user can query):

QS	Error Magnitude (pulses)

User Configurable Commands (user can query & change):

OE	Profiler Off-On Error
YA	Step Drive Resolution (pulses / full motor step)
YB	Step Motor Resolution (full motor steps / revolution)
YC	Encoder Resolution (counts / revolution)
YR	Error Correction (pulses)
YS	Stepper Position Maintenance enable, status

A pulse is defined by the resolution of the step drive being used. Therefore, one pulse could be a full step, a half step or a microstep.

When a Galil controller is configured for step motor operation, the step pulse output by the controller is internally fed back to the auxiliary encoder register. For SPM the feedback encoder on the stepper will connect to the main encoder port. Enabling the SPM mode on a controller with YS=1 executes an internal monitoring of the auxiliary and main encoder registers for that axis or axes. Position error is then tracked in step pulses between these two registers (QS command).

$$QS = TD - \frac{TP \times YA \times YB}{YC}$$

Where TD is the auxiliary encoder register(step pulses) and TP is the main encoder register(feedback encoder). Additionally, YA defines the step drive resolution where YA = 1 for full stepping or YA = 2 for half stepping. The full range of YA is up to YA = 9999 for microstepping drives.

Error Limit

The value of QS is internally monitored to determine if it exceeds a preset limit of three full motor steps. Once the value of QS exceeds this limit, the controller then performs the following actions:

1. The motion is maintained or is stopped, depending on the setting of the OE command. If OEA=0 the axis stays in motion, if OEA=1 the axis is stopped.
2. YS is set to 2, which causes the automatic subroutine labeled #POSERR to be executed.

Correction

A correction move can be commanded by assigning the value of QS to the YR correction move command. The correction move is issued only after the axis has been stopped. After an error correction move has completed and

QS is less than three full motor steps, the YS error status bit is automatically reset back to 1; indicating a cleared error.

Example: SPM Mode Setup

The following code demonstrates what is necessary to set up SPM mode for a full step drive, a half step drive, and a 1/64th microstepping drive for an axis with a 1.8° step motor and 4000 count/rev encoder. Note the necessary difference is with the YA command.

Full-Stepping Drive, X axis:

```
      #SETUP
      OE1;        Set the profiler to stop axis upon error
      KS16        Set step smoothing;
      MT-2;       Motor type set to stepper
      YA1;        Step resolution of the full-step drive
      YB200;      Motor resolution (full steps per revolution)
      YC4000;     Encoder resolution (counts per revolution)
      SHX;        Enable axis
      WT50;       Allow slight settle time
      YS1;        Enable SPM mode
```

Half-Stepping Drive, X axis:

```
      #SETUP
      OE1;        Set the profiler to stop axis upon error
      KS16        Set step smoothing;
      MT-2;       Motor type set to stepper
      YA2;        Step resolution of the half-step drive
      YB200;      Motor resolution (full steps per revolution)
      YC4000;     Encoder resolution (counts per revolution)
      SHX;        Enable axis
      WT50;       Allow slight settle time
      YS1;        Enable SPM mode
```

1/64th Step Microstepping Drive, X axis:

```
      #SET
UP
      OE1;       Set the profiler to stop axis upon
           error
      KS16       Set step smoothing
;
      MT-        Motor type set to stepper
2;
      YA64       Step resolution of the
;          microstepping drive
      YB20       Motor resolution (full steps per
0;         revolution)
      YC40       Encoder resolution (counts per
00;        revolution)
      SHX;       Enable axis
      WT50       Allow slight settle time
;
      YS1;       Enable SPM mode
```

Example: Error Correction

The following code demonstrates what is necessary to set up SPM mode in order to detect the error, stop the motor, correct the error, and return to the main code. The drive is a full step drive, with a 1.8º step motor and 4000 count/rev encoder.

```
      #SETUP
      OE1;             Set the profiler to stop axis upon error
      KS16;            Set step smoothing
      MT-2;            Motor type set to stepper
      YA2;             Step resolution of the drive
      YB200;           Motor resolution (full steps per revolution)
      YC4000;          Encoder resolution (counts per revolution)
      SHX;             Enable axis
      WT100;           Allow slight settle time


      #MOTION          Perform motion
      SP512;           Set the speed
      PR1000;          Prepare mode of motion
      BGX;             Begin motion
      #LOOP;JP#L       Keep thread zero alive for #POSERR to run in
OOP;
```

```
      REM  When error occurs, the axis will stop due to OE1.  In
      REM #POSERR, query the status YS and the error QS, correct,
      REM and return to the main code.

      #POSERR;          Automatic subroutine is called when YS=2
      WT100;            Wait helps user see the correction
      spsave=_SP        Save current speed setting
X;
      JP#RETURN,        Return to thread zero if invalid error
_YSX<>2;
      SP64;             Set slow speed setting for correction
      MG"ERROR=
",_QSX
      YRX=_QSX;         Else, error is valid, use QS for correction
      MCX;              Wait for motion to complete
      MG"CORRECTED, ERROR NOW= ",_QSX
      WT100;            Wait helps user see the correction

      #RETURN
      SPX=spsave        Return the speed to previous setting
;
      RE0;              Return from #POSERR
```

Example: Friction Correction

The following example illustrates how the SPM mode can be useful in correcting for X axis friction after each move when conducting a reciprocating motion. The drive is a 1/256th microstepping drive with a 1.8° step motor and 4000 count/rev encoder.

```
      #SETUP;           Set the profiler to continue upon error
      KS16;             Set step smoothing
      MT-2;             Motor type set to stepper
      YA256;            Step resolution of the microstepping drive
      YB200;            Motor resolution (full steps per revolution)
      YC4000;           Encoder resolution (counts per revolution)
      SHX;              Enable axis
      WT50;             Allow slight settle time
      YS1;              Enable SPM mode

      #MOTION;                    Perform motion
      SP16384;                    Set the speed
      PR10000;                    Prepare mode of motion
      BGX;                        Begin motion
      MCX
      JS#CORRECT;                 Move to correction
      #MOTION2
```

```
SP16384;                        Set the speed
PR-10000;                       Prepare mode of motion
BGX;                            Begin motion
MCX
JS#CORRECT;                     Move to correction
JP#MOTION
#CORRECT;                       Correction code
spx=_SPX
#LOOP;                          Save speed value
SP2048;                         Set a new slow correction speed
WT100;                          Stabilize
JP#END,@ABS[_QSX]<10;           End correction if error is
                              within defined tolerance
YRX=_QSX;                       Correction move
MCX
WT100;                          Stabilize
JP#LOOP;                        Keep correcting until error is
                              within tolerance
#END;                           End #CORRECT subroutine,
                              returning to code
SPX=spx
EN
```

Dual Loop (Auxiliary Encoder)

The DMC-30000 provides an interface for a second encoder except when the controller is configured for stepper motor operation or used in circular compare. When used, the second encoder is typically mounted on the motor or the load, but may be mounted in any position. The most common use for the second encoder is backlash compensation, described below.

The second encoder may be a standard quadrature type, or it may provide pulse and direction. The controller also offers the provision for inverting the direction of the encoder rotation. The main and the auxiliary encoders are configured with the CE command. The command form is CE x where the parameter x is the sum of two integers m and n. m configures the main encoder and n configures the auxiliary encoder.

Using the CE Command

m=	Main Encoder	n=	Second Encoder
0	Normal quadrature	0	Normal quadrature
1	Pulse & direction	4	Pulse & direction
2	Reverse quadrature	8	Reversed quadrature
3	Reverse pulse & direction	12	Reversed pulse & direction

For example, to configure the main encoder for reversed quadrature, m=2, and a second encoder of pulse and direction, n=4, the total is 6, and the command for the X axis is:

```
CE 6
```

Additional Commands for the Auxiliary Encoder

The command, DE, can be used to define the position of the auxiliary encoders. For example,

```
DE 500
```

sets the value to 500. The positions of the auxiliary encoders may be interrogated with the command, DE? or the operand _DEA.

The command TD X returns the current position of the auxiliary encoder.

The command DV 1 configures the auxiliary encoder to be used for backlash compensation.

Backlash Compensation

There are two methods for backlash compensation using the auxiliary encoders:

1. Continuous dual loop
2. Sampled dual loop

To illustrate the problem, consider a situation in which the coupling between the motor and the load has a backlash. To compensate for the backlash, position encoders are mounted on both the motor and the load.

The continuous dual loop combines the two feedback signals to achieve stability. This method requires careful system tuning, and depends on the magnitude of the backlash. However, once successful, this method compensates for the backlash continuously.

The second method, the sampled dual loop, reads the load encoder only at the end point and performs a correction. This method is independent of the size of the backlash. However, it is effective only in point-to-point motion systems which require position accuracy only at the endpoint.

Continuous Dual Loop - Example

Connect the load encoder to the main encoder port and connect the motor encoder to the dual encoder port. The dual loop method splits the filter function between the two encoders. It applies the KP (proportional) and KI (integral) terms to the position error, based on the load encoder, and applies the KD (derivative) term to the motor encoder. This method results in a stable system.

The dual loop method is activated with the instruction DV (Dual Velocity), where

```
DV        1
```

activates dual loop and

```
DV        0
```

disables dual loop.

NOTE: Dual loop compensation depends on the backlash magnitude, and in extreme cases will not stabilize the loop. The proposed compensation procedure is to start with KPA=0, KIA=0 and to maximize the value of KD under the condition DV1. Once KD is found, increase KP gradually to a maximum value, and finally, increase KI, if necessary.

Sampled Dual Loop - Example

In this example, we consider a linear slide which is run by a rotary motor via a lead screw. Since the lead screw has a backlash, it is necessary to use a linear encoder to monitor the position of the slide. For stability reasons, it is best to use a rotary encoder on the motor.

Connect the rotary encoder to the X-axis and connect the linear encoder to the auxiliary encoder of X. Assume that the required motion distance is one inch, and that this corresponds to 40,000 counts of the rotary encoder and 10,000 counts of the linear encoder.

The design approach is to drive the motor a distance, which corresponds to 40,000 rotary counts. Once the motion is complete, the controller monitors the position of the linear encoder and performs position corrections.

This is done by the following program.

```
INSTRUCTION          INTERPRETATION
      #DUALOOP           Label
      CE 0               Configure encoder
      DE0                Set initial value
      PR 40000           Main move
      BGX                Start motion
      #Correct           Correction loop
      AMX                Wait for motion completion
      V1=10000-          Find linear encoder error
_DEX
      V2=-               Compensate for motor error
_TEX/4+V1
      JP#END,@ABS[       Exit if error is small
V2]<2
      PR V2*4            Correction move
      BGX                Start correction
      JP#CORRECT         Repeat
      #END
      EN
```

Motion Smoothing

The DMC-30000 controller allows the smoothing of the velocity profile to reduce the mechanical vibration of the system.

Trapezoidal velocity profiles have acceleration rates which change abruptly from zero to maximum value. The discontinuous acceleration results in jerk which causes vibration. The smoothing of the acceleration profile leads to a continuous acceleration profile and reduces the mechanical shock and vibration.

Using the IT Command:

When operating with servo motors, motion smoothing can be accomplished with the IT command. This command filters the acceleration and deceleration functions to produce a smooth velocity profile. The resulting velocity profile, has continuous acceleration and results in reduced mechanical vibrations.

The smoothing function is specified by the following command:

IT x	Independent time constant

The command, IT, is used for smoothing independent moves of the type JG, PR, PA and to smooth vector moves of the type VM and LM.

The smoothing parameter, x, is a number between 0 and 1 and determine the degree of filtering. The maximum value of 1 implies no filtering, resulting in trapezoidal velocity profiles. Smaller values of the smoothing parameters imply heavier filtering and smoother moves.

The following example illustrates the effect of smoothing. Figure 6.13 shows the trapezoidal velocity profile and the modified acceleration and velocity.

Note that the smoothing process results in longer motion time.

Example - Smoothing

```
     PR          Position
20000
     AC          Acceleration
100000
     DC          Deceleration
100000
     SP          Speed
5000
     IT .5       Filter for smoothing
     BG X        Begin
```

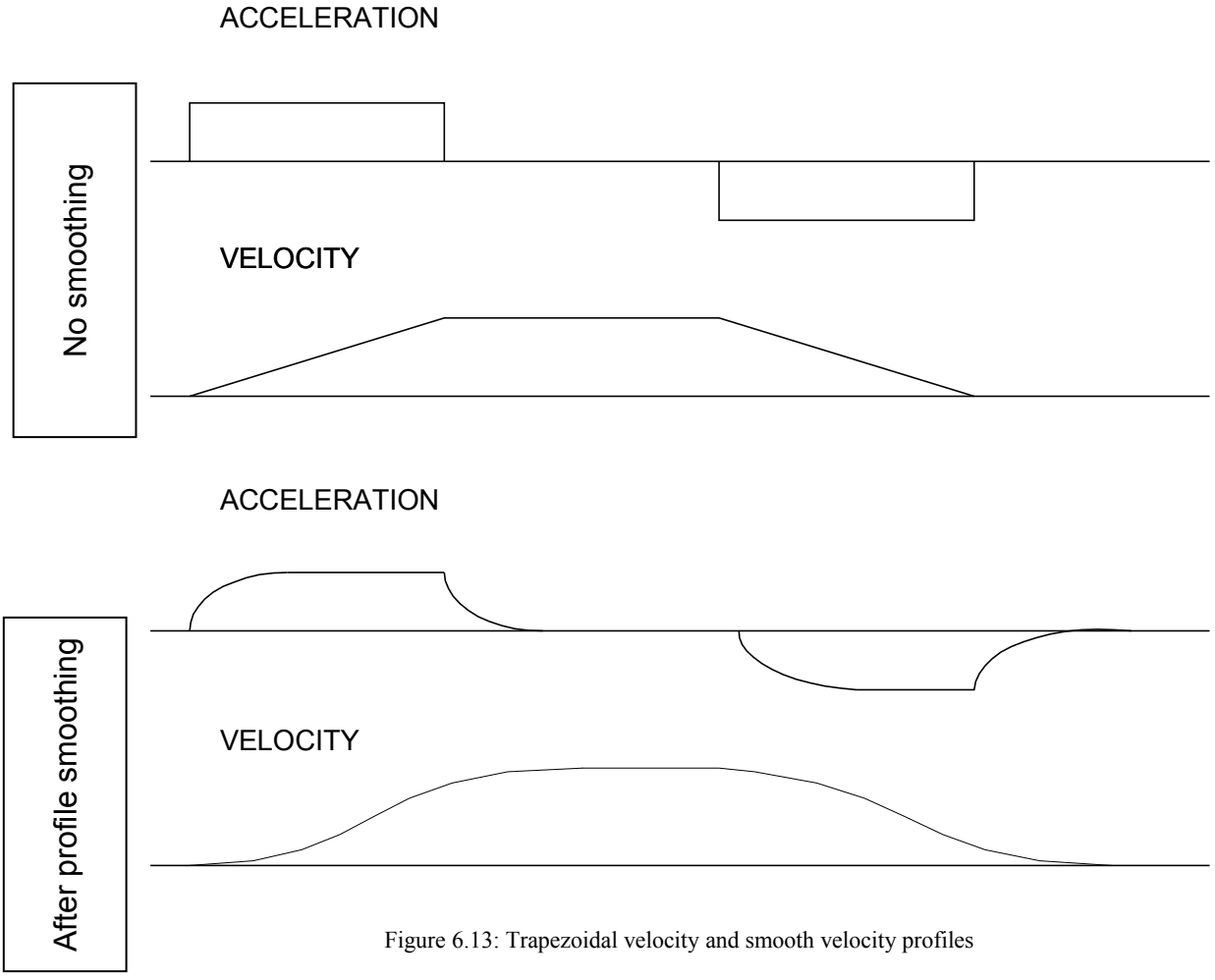


Figure 6.13: Trapezoidal velocity and smooth velocity profiles

Using the KS Command (Step Motor Smoothing):

When operating with step motors, motion smoothing can be accomplished with the command, KS. The KS command smoothes the frequency of step motor pulses. Similar to the command IT, this produces a smooth velocity profile.

The step motor smoothing is specified by the following command:

KS x where x is an integer from 0.5 to 128 and represents the amount of smoothing

The smoothing parameters, x,y,z,w and n are numbers between 0.5 and 128 and determine the degree of filtering. The minimum value of 0.5 implies the least filtering, resulting in trapezoidal velocity profiles. Larger values of the smoothing parameters imply heavier filtering and smoother moves.

Note that KS is valid only for step motors.

Homing

The Find Edge (FE) and Home (HM) instructions may be used to home the motor to a mechanical reference. This reference is connected to the Home input line. The HM command initializes the motor to the encoder index pulse in addition to the Home input. The configure command (CN) is used to define the polarity of the home input.

The Find Edge (FE) instruction is useful for initializing the motor to a home switch. The home switch is connected to the Homing Input. When the Find Edge command and Begin is used, the motor will accelerate up to the slew speed and slew until a transition is detected on the Homing line. The motor will then decelerate to a stop. A high deceleration value must be input before the find edge command is issued for the motor to decelerate rapidly after sensing the home switch. The Home (HM) command can be used to position the motor on the index pulse after the home switch is detected. This allows for finer positioning on initialization. The HM command and BG command causes the following sequence of events to occur.

Stage 1:

Upon begin, the motor accelerates to the slew speed specified by the JG or SP commands. The direction of its motion is determined by the state of the homing input. If _HMX reads 1 initially, the motor will go in the reverse direction first (direction of decreasing encoder counts). If _HMX reads 0 initially, the motor will go in the forward direction first. CN is the command used to define the polarity of the home input. With CN,-1 (the default value) a normally open switch will make _HMX read 1 initially, and a normally closed switch will make _HMX read zero. Furthermore, with CN,1 a normally open switch will make _HMX read 0 initially, and a normally closed switch will make _HMX read 1. Therefore, the CN command will need to be configured properly to ensure the correct direction of motion in the home sequence.

Upon detecting the home switch changing state, the motor begins decelerating to a stop.

NOTE: The direction of motion for the FE command also follows these rules for the state of the home input.

Stage 2:

The motor then traverses at HV counts/sec in the opposite direction of Stage 1 until the home switch toggles again. If Stage 3 is in the opposite direction of Stage 2, the motor will stop immediately at this point and change direction. If Stage 2 is in the same direction as Stage 3, the motor will never stop, but will smoothly continue into Stage 3.

Stage 3:

The motor traverses forward at HV counts/sec until the encoder index pulse is detected. The motor then decelerates to a stop and goes back to the index.

The DMC-30000 defines the home position as the position at which the index was detected and sets the encoder reading at this point to zero.

The 4 different motion possibilities for the home sequence are shown in the following table.

			Direction of Motion		
Switch Type	CN Setting	Initial _HMX state	Stage 1	Stage 2	Stage 3
Normally Open	CN,-1	1	Reverse	Forward	Forward
Normally Open	CN,1	0	Forward	Reverse	Forward
Normally Closed	CN,-1	0	Forward	Reverse	Forward
Normally Closed	CN,1	1	Reverse	Forward	Forward

Example: Homing

```
Instruction              Interpretation
      #HOME              Label
      CN,-1              Configure the polarity of the home input
      AC 1000000         Acceleration Rate
      DC 1000000         Deceleration Rate
      SP 5000            Speed for Home Search
      HM                 Home
      BG                 Begin Motion
      AM                  After Complete
      MG "AT             Send Message
HOME"
      EN                 End
```

Figure 6.14 shows the velocity profile from the homing sequence of the example program above. For this profile, the switch is normally closed and CN,-1.

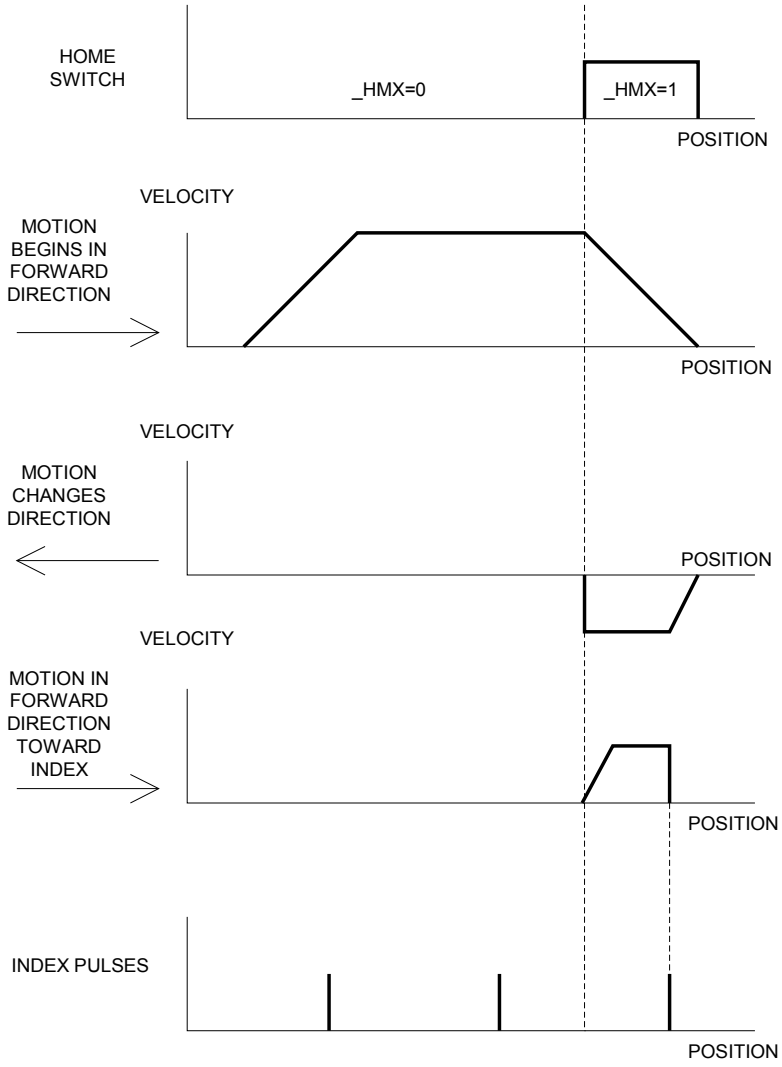


Figure 6.14: Homing Sequence for Normally Closed Switch and CN,-1

Example: Find Edge

```
#EDGE              Label
AC 2000000         Acceleration rate
DC 2000000         Deceleration rate
SP 8000            Speed
FE                 Find edge command
BG                 Begin motion
AM                 After complete
MG "FOUND          Send message
HOME"
DP 0               Define position as 0
EN                 End
```

Command Summary - Homing Operation

command	description
FE A	Find Edge Routine. This routine monitors the Home Input
FI A	Find Index Routine - This routine monitors the Index Input
HM A	Home Routine - This routine combines FE and FI as Described Above
SC A	Stop Code
TS A	Tell Status of Switches and Inputs

Operand Summary - Homing Operation

operand	Description
_HMA	Contains the value of the state of the Home Input
_SCA	Contains stop code
_TSA	Contains status of switches and inputs

High Speed Position Capture (The Latch Function)

Often it is desirable to capture the position precisely for registration applications. The DMC-30000 provides a position latch feature. This feature allows the position of the main or auxiliary encoders of X,Y,Z or W to be captured within 25 microseconds of an external low input signal (or index pulse). General inputs 1 is the latch input.

NOTE: To insure a position capture within 25 microseconds, the input signal must be a transition from high to low.

The DMC-30000 software commands, AL and RL, are used to arm the latch and report the latched position. The steps to use the latch are as follows:

1. Give the AL A to arm the latch for the main encoder and ALSA for the auxiliary encoder.
2. Test to see if the latch has occurred (Input goes low) by using the _ALX command. Example, V1=_ALX returns the state of the latch into V1. V1 is 1 if the latch has not occurred.
3. After the latch has occurred, read the captured position with the RLX command or _RLX.

NOTE: The latch must be re-armed after each latching event.

Example:

```
      #Latch       Latch program
      JG           Jog
5000
      BG           Begin motion
      AL A         Arm Latch
      #Wait        #Wait label for loop
      JP           Jump to #Wait label if latch has not occurred
#Wait,_ALA=1
      Result       Set value of variable 'Result' equal to the report
=_RLA         position
      MG           Print result
Result
      EN           End
```

Real Time Clock

The DMC-30000 is equipped with a real time clock feature. The real time clock provides true time in seconds, minutes and hours. The RT command provides a method to set the time and operands to return the current time. The default real time clock does not persist through a power cycle and must be set whenever power is restored.

The DMC-30000 can be ordered with a clock upgrade (RTC) including a higher precision clock than the default, and a battery backup for the time hardware. All hardware is within the standard sheet metal footprint. The RTC clock will continue to run when power is removed from the cotnroller. The RTC option also provides a calendar function including year, month of year, day of month, and day of week. This feature can be set and queried through the RY command.

Both versions of the real time clock can be set to a TIME protocol (RFC 868) server. Using IH, the DMC-30000 can connect to a TIME server over TCP on port 37 and receive the 32bit response. The firmware will then set the time and calendar (if applicable) to the TIME server value. The command RO is used to set the GMT time zone offset for localization of the current time. The TIME protocol synchronization is designed to connect to a server on the local network. Contact Galil if a local server is not available (e.g. an Internet Gateway is required to contact NIST).

See the Error: Reference source not found section in the Appendix for further details and specifications for the RTC option.

Chapter 7 Application Programming

Overview

The DMC-30000 provides a powerful programming language that allows users to customize the controller for their particular application. Programs can be downloaded into the DMC-30000 memory freeing the host computer for other tasks. However, the host computer can send commands to the controller at any time, even while a program is being executed. Only ASCII commands can be used for application programming.

In addition to standard motion commands, the DMC-30000 provides commands that allow the DMC-30000 to make its own decisions. These commands include conditional jumps, event triggers and subroutines. For example, the command JP#LOOP, n<10 causes a jump to the label #LOOP if the variable n is less than 10.

For greater programming flexibility, the DMC-30000 provides user-defined variables, arrays and arithmetic functions. For example, with a cut-to-length operation, the length can be specified as a variable in a program which the operator can change as necessary.

The following sections in this chapter discuss all aspects of creating applications programs. The program memory size is 40 characters x 1000 lines.

Program Format

A DMC-30000 program consists of DMC instructions combined to solve a machine control application. Action instructions, such as starting and stopping motion, are combined with Program Flow instructions to form the complete program. Program Flow instructions evaluate real-time conditions, such as elapsed time or motion complete, and alter program flow accordingly.

Each DMC-30000 instruction in a program must be separated by a delimiter. Valid delimiters are the semicolon (;) or carriage return. The semicolon is used to separate multiple instructions on a single program line where the maximum number of instructions on a line is limited by 80 characters. A carriage return enters the final command on a program line.

Using Labels in Programs

All DMC-30000 programs must begin with a label and end with an End (EN) statement. Labels start with the pound (#) sign followed by a maximum of seven characters. The first character must be a letter; after that, numbers are permitted. Spaces are not permitted in a label.

The maximum number of labels which may be defined is 510.

Valid labels

```
#BEGIN
#SQUARE
#X1
```

```
#BEGIN1
```

Invalid labels

```
#1Square
#123
```

A Simple Example Program:

```
#START          Beginning of the Program
PR 10000        Specify relative distance
BG A            Begin Motion
AM A            Wait for motion complete
WT 2000         Wait 2 sec
JP #START       Jump to label START
EN              End of Program
```

The above program moves 10000 counts. After the motion is complete, the motors rest for 2 seconds. The cycle repeats indefinitely until the stop command is issued.

Special Labels

The DMC-30000 have some special labels, which are used to define input interrupt subroutines, limit switch subroutines, error handling subroutines, and command error subroutines. See section on ___

#AMPERR	Label for Amplifier error routine
#AUTO	Label that will automatically run upon the controller exiting a reset (power-on)
#AUTOERR	Label that will automatically run if there is an FLASH error out of reset
#CMDERR	Label for incorrect command subroutine
#COMINT	Label for Communications Interrupt (See CC Command)
#ININT	Label for Input Interrupt subroutine (See II Command)
#LIMSWI	Label for Limit Switch subroutine
#MCTIME	Label for timeout on Motion Complete trip point
#POSERR	Label for excess Position Error subroutine
#TCPERR	Label for errors over a TCP connection (error code 123)

Commenting Programs

Using the command, NO or Apostrophe (‘)

The DMC-30000 provides a command, NO, for commenting programs or single apostrophe. This command allows the user to include up to 38 characters on a single line after the NO command and can be used to include comments from the programmer.

NOTE: The NO and (') commands are actual controller commands. Therefore, inclusion of the NO or (') commands will require process time by the controller, see General Program Flow and Timing information for more details.

Difference between NO and ' using the GalilTools software

The GalilTools software will treat an apostrophe (') commend different from an NO when the compression algorithm is activated upon a program download (line > 40 characters or program memory > 1000 lines). In this case the software will remove all (') comments as part of the compression and it will download all NO comments to the controller.

Executing Programs - Multitasking

The DMC-30000 can run up to 4 independent programs simultaneously. These programs are called threads and are numbered 0 through 3, where 0 is the main thread. Multitasking is useful for executing independent operations such as PLC functions that occur independently of motion.

The main thread differs from the others in the following ways:

1. When input interrupts are implemented for limit switches, position errors or command errors, the subroutines are executed as thread 0.

To begin execution of the various programs, use the following instruction:

```
XQ #A, n
```

Where n indicates the thread number. To halt the execution of any thread, use the instruction

```
HX n
```

where n is the thread number.

Note that both the XQ and HX commands can be performed by an executing program.

The example below produces a waveform on Output 1 independent of a move.

```
      #TASK1            Task1 label
      AT0               Initialize reference time
      CB1               Clear Output 1
      #LOOP1            Loop1 label
      AT 10             Wait 10 msec from reference time
      SB1               Set Output 1
      AT -40            Wait 40 msec from reference time, then
                  initialize reference
      CB1               Clear Output 1
      JP #LOOP1         Repeat Loop1
      #TASK2            Task2 label
      XQ #TASK1,1       Execute Task1
      #LOOP2            Loop2 label
      PR 1000           Define relative distance
      BGX               Begin motion
      AMX               After motion done
      WT 10             Wait 10 msec
      JP                Repeat motion unless Input 2 is low
#LOOP2,@IN[2]=1
      HX                Halt all tasks
```

The program above is executed with the instruction XQ #TASK2,0 which designates TASK2 as the main thread (i.e. Thread 0). #TASK1 is executed within TASK2.

Debugging Programs

The DMC-30000 provides commands and operands which are useful in debugging application programs. These commands include interrogation commands to monitor program execution, determine the state of the controller and

the contents of the controllers program, array, and variable space. Operands also contain important status information which can help to debug a program.

Trace Commands

The trace command causes the controller to send each line in a program to the host computer immediately prior to execution. Tracing is enabled with the command, TR1. TR0 turns the trace function off. Note: When the trace function is enabled, the line numbers as well as the command line will be displayed as each command line is executed.

NOTE: When the trace function is enabled, the line numbers as well as the command line will be displayed as each command line is executed.

Error Code Command

When there is a program error, the DMC-30000 halts the program execution at the point where the error occurs. To display the last line number of program execution, issue the command, MG _ED.

The user can obtain information about the type of error condition that occurred by using the command, TC1. This command reports back a number and a text message which describes the error condition. The command, TC0 or TC, will return the error code without the text message. For more information about the command, TC, see the Command Reference.

Stop Code Command

The status of motion for each axis can be determined by using the stop code command, SC. This can be useful when motion on an axis has stopped unexpectedly. The command SC will return a number representing the motion status. See the command reference for further information.

Flash Memory Interrogation Commands

For debugging the status of the program memory, array memory, or variable memory, the DMC-30000 has several useful commands. The command, DM ?, will return the number of array elements currently available. The command, DA ?, will return the number of arrays which can be currently defined. For example, a standard DMC-30000 will have a maximum of 3000 array elements in up to 6 arrays. If an array of 100 elements is defined, the command DM ? will return the value 2900 and the command DA ? will return 5.

To list the contents of the variable space, use the interrogation command LV (List Variables). To list the contents of array space, use the interrogation command, LA (List Arrays). To list the contents of the Program space, use the interrogation command, LS (List). To list the application program labels only, use the interrogation command, LL (List Labels).

Operands

In general, all operands provide information which may be useful in debugging an application program. Below is a list of operands which are particularly valuable for program debugging. To display the value of an operand, the message command may be used. For example, since the operand, _ED contains the last line of program execution, the command MG _ED will display this line number.

_ED contains the last line of program execution. Useful to determine where program stopped.

_DL contains the number of available labels.

_UL contains the number of available variables.

_DA contains the number of available arrays.

_DM contains the number of available array elements.

_AB contains the state of the Abort Input

_LFx contains the state of the forward limit switch for the ‘x’ axis

_LRx contains the state of the reverse limit switch for the ‘x’ axis

Debugging Example:

The following program has an error. It attempts to specify a relative movement while the X-axis is already in motion. When the program is executed, the controller stops at line 003. The user can then query the controller using the command, TC1. The controller responds with the corresponding explanation:

```
      Download
Code
      #A                Program Label
      PR1000            Position Relative 1000
      BGX               Begin
      PR5000            Position Relative 5000
      EN                End
      From
Terminal
      :XQ #A            Execute #A
      ?003 PR5000       Error on Line 3
      :TC1              Tell Error Code
      ?7 Command        Command not valid while running
not valid while
running.
                  Change the BGX line to BGX;AMX and re-download
                 the program.
      :XQ #A            Execute #A
```

Program Flow Commands

The DMC-30000 provides instructions to control program flow. The controller program sequencer normally executes program instructions sequentially. The program flow can be altered with the use of event triggers, trippoints, and conditional jump statements.

Event Triggers & Trippoints

To function independently from the host computer, the DMC-30000 can be programmed to make decisions based on the occurrence of an event. Such events include waiting for motion to be complete, waiting for a specified amount of time to elapse, or waiting for an input to change logic levels.

The DMC-30000 provides several event triggers that cause the program sequencer to halt until the specified event occurs. Normally, a program is automatically executed sequentially one line at a time. When an event trigger instruction is decoded, however, the actual program sequence is halted. The program sequence does not continue until the event trigger is "tripped". For example, the motion complete trigger can be used to separate two move sequences in a program. The commands for the second move sequence will not be executed until the motion is complete on the first motion sequence. In this way, the controller can make decisions based on its own status or external events without intervention from a host computer.

DMC-30000 Event Triggers

Command	Function

AM A or S	Halts program execution until motion is complete on the specified axes or motion sequence(s). This command is useful for separating motion sequences in a program.
AD A	Halts program execution until position command has reached the specified relative distance from the start of the move.
AR A	Halts program execution until after specified distance from the last AR or AD command has elapsed.
AP A	Halts program execution until after absolute position occurs.
MF A	Halt program execution until after forward motion reached absolute position. If position is already past the point, then MF will trip immediately. Will function on geared axis or aux. inputs.
MR A	Halt program execution until after reverse motion reached absolute position. If position is already past the point, then MR will trip immediately. Will function on geared axis or aux. inputs.
MC A	Halt program execution until after the motion profile has been completed and the encoder has entered or passed the specified position. TW x sets timeout to declare an error if not in position. If timeout occurs, then the trippoint will clear and the stop code will be set to 99. An application program will jump to label #MCTIME.
AI +/- n	Halts program execution until after specified input is at specified logic level. n specifies input line. Positive is high logic level, negative is low level. n=1 through 8
AS A	Halts program execution until the axis has reached its slew speed.
AT +/-n,m	For m=omitted or 0, halts program execution until n msec from reference time. AT 0 sets reference. AT n waits n msec from reference. AT -n waits n msec from reference and sets new reference after elapsed time. For m=1. Same functionality except that n is number of samples rather than msec
AV n	Halts program execution until specified distance along a coordinated path has occurred.
WT n,m	For m=omitted or 0, halts program execution until specified time in msec has elapsed. For m=1. Same functionality except that n is number of samples rather than msec.

Event Trigger Examples:

Event Trigger - Multiple Move Sequence

The AM trippoint is used to separate the two PR moves. If AM is not used, the controller returns a ? for the second PR command because a new PR cannot be given until motion is complete.

```
#TWOMOVE;'        Label
PR 2000;'         Position Command
BGX;'             Begin Motion
```

```
AMX;'            Wait for Motion Complete
PR 4000;'        Next Position Move
BGX;'            Begin 2nd move
EN;'             End program
```

Event Trigger - Set Output after Distance

Set output bit 1 after a distance of 1000 counts from the start of the move. The accuracy of the trippoint is the speed multiplied by the sample period.

```
#SETBIT;'        Label
SP 10000;'       Speed is 10000
PA 20000;'       Specify Absolute position
BGX;'            Begin motion
AD 1000;'        Wait until 1000 counts
SB1;'            Set output bit 1
EN;'             End program
```

Event Trigger - Repetitive Position Trigger

To set the output bit every 10000 counts during a move, the AR trippoint is used as shown in the next example.

```
#TRIP;'          Label
JG 50000;'       Specify Jog Speed
BGX;n=0;'        Begin Motion
#REPEAT;'        # Repeat Loop
AR 10000;'       Wait 10000 counts
TPX;'            Tell Position
SB1;'            Set output 1
WT50;'           Wait 50 msec
CB1;'            Clear output 1
n=n+1;'          Increment counter
JP               Repeat 5 times
#REPEAT,n<5;'
STX;'            Stop
EN;'             End
```

Event Trigger - Start Motion on Input

This example waits for input 1 to go low and then starts motion. Note: The AI command actually halts execution of the program until the input occurs. If you do not want to halt the program sequences, you can use the Input Interrupt function (II) or use a conditional jump on an input, such as `JP#GO,@IN[1] = 1`.

```
#INPUT;'         Program Label
AI-1;'           Wait for input 1 low
PR 10000;'       Position command
BGX;'            Begin motion
EN;'             End program
```

Event Trigger - Set output when At speed

```
#ATSPEED;'        Program Label
JG 50000;'        Specify jog speed
AC 10000;'        Acceleration rate
BGX;'             Begin motion
ASX;'             Wait for at slew speed 50000
SB1;'             Set output 1
EN;'              End program
```

Event Trigger - Change Speed along Vector Path

The following program changes the feed rate or vector speed at the specified distance along the vector. The vector distance is measured from the start of the move or from the last AV command.

```
#VECTOR;'         Label
VM XN;VS          Coordinated path
5000;'
VP                Vector position
10000,20000;'
VP                Vector position
20000,30000;'
VE;'              End vector
BGS;'             Begin sequence
AV 5000;'         After vector distance
VS 1000;'         Reduce speed
EN;'              End
```

Event Trigger - Multiple Move with Wait

This example makes multiple relative distance moves by waiting for each to be complete before executing new moves.

```
#MOVES;'          Label
PR 12000;'        Distance
SP 20000;'        Speed
AC 100000;'       Acceleration
BGX;'             Start Motion
AD 10000;'        Wait a distance of 10,000 counts
SP 5000;'         New Speed
AMX;'             Wait until motion is completed
WT 200;'          Wait 200 ms
PR -10000;'       New Position
SP 30000;'        New Speed
AC 150000;'       New Acceleration
BGX;'             Start Motion
EN;'              End
```

Define Output Waveform Using AT

The following program causes Output 1 to be high for 10 msec and low for 40 msec. The cycle repeats every 50 msec.

```
#OUTPUT;'          Program label
AT0;'              Initialize time reference
SB1;'              Set Output 1
#LOOP;'            Loop
AT 10;'            After 10 msec from reference,
CB1;'              Clear Output 1
AT -40;'           Wait 40 msec from reference and reset
                reference
SB1;'              Set Output 1
JP #LOOP;'         Loop
EN;'               End Program
```

Using AT/WT with non-default TM rates

By default both WT and AT are defined to hold up program execution for 'n' number of milliseconds (WT n or AT n). The second field of both AT and WT can be used to have the program execution be held-up for 'n' number of samples rather than milliseconds. For example WT 400 or WT 400,0 will hold up program execution for 400 msec regardless of what is set for TM. By contrast WT 400,1 will hold up program execution for 400 samples. For the default TM of 1000 the servo update rate is 976us per sample, so the difference between WT n,0 and WT n,1 is minimal. The difference comes when the servo update rate is changed. With a low servo update rate, it is often useful to be able to time loops based upon samples rather than msec, and this is where the “unscaled” WT and AT are useful. For example:

```
#MAIN;'            Label
TM 250;'           250us update rate
#MOVE;'            Label
PRX=1000;'         Position Relative Move
BGX;'              Begin Motion
MCX;'              Wait for motion to complete
WT 2,1;'           Wait 2 samples (500us)
SB1;'              Set bit 1
EN;'               End Program
```

In the above example, without using an unscaled WT, the output would either need to be set directly after the motion was complete, or 2 ms after the motion was complete. By using WT n,1 and a lower TM, greater delay resolution was achieved.

Conditional Jumps

The DMC-30000 provides Conditional Jump (JP) and Conditional Jump to Subroutine (JS) instructions for branching to a new program location based on a specified condition. The conditional jump determines if a condition is satisfied and then branches to a new location or subroutine. Unlike event triggers, the conditional jump instruction does not halt the program sequence. Conditional jumps are useful for testing events in real-time. They allow the controller to make decisions without a host computer. For example, the DMC-30000 can decide between two motion profiles based on the state of an input line.

Command Format - JP and JS

FORMAT:	DESCRIPTION
JS destination, logical condition	Jump to subroutine if logical condition is satisfied
JP destination, logical condition	Jump to location if logical condition is satisfied

The destination is a program line number or label where the program sequencer will jump if the specified condition is satisfied. Note that the line number of the first line of program memory is 0. The comma designates "IF". The logical condition tests two operands with logical operators.

Logical operators:

OPERATOR	DESCRIPTION
<	less than
>	greater than
=	equal to
<=	less than or equal to
>=	greater than or equal to
<>	not equal

Conditional Statements

The conditional statement is satisfied if it evaluates to any value other than zero. The conditional statement can be any valid DMC-30000 numeric operand, including variables, array elements, numeric values, functions, keywords, and arithmetic expressions. If no conditional statement is given, the jump will always occur.

Examples:

Number	`v1=6`
Numeric Expression	`v1=v7*6`
	`@ABS[v1]>10`
Array Element	`v1<count[2]`
Variable	`v1<v2`
Internal Variable	`_TPX=0`
	`_TVX>500`
I/O	`v1>@AN[2]`
	`@IN[1]=0`

Multiple Conditional Statements

The DMC-30000 will accept multiple conditions in a single jump statement. The conditional statements are combined in pairs using the operands "&" and "|". The "&" operand between any two conditions, requires that both statements must be true for the combined statement to be true. The "|" operand between any two conditions, requires that only one statement be true for the combined statement to be true.

NOTE: Each condition must be placed in parentheses for proper evaluation by the controller. In addition, the DMC-30000 executes operations from left to right. See Mathematical and Functional Expressions for more information.

For example, using variables named v1, v2, v3 and v4:

```
JP #TEST,((v1<v2)&(v3<v4))
```

In this example, this statement will cause the program to jump to the label #TEST if v1 is less than v2 and v3 is less than v4. To illustrate this further, consider this same example with an additional condition:

```
JP #TEST, ((v1<v2) & (v3<v4)) | (v5<v6)
```

This statement will cause the program to jump to the label #TEST under two conditions; 1. If v1 is less than v2 and v3 is less than v4. OR 2. If v5 is less than v6.

Using the JP Command:

If the condition for the JP command is satisfied, the controller branches to the specified label or line number and continues executing commands from this point. If the condition is not satisfied, the controller continues to execute the next commands in sequence.

Conditional	Meaning
JP #Loop,count<10	Jump to #Loop if the variable, count, is less than 10
JS #MOVE2,@IN[1]=1	Jump to subroutine #MOVE2 if input 1 is logic level high. After the subroutine MOVE2 is executed, the program sequencer returns to the main program location where the subroutine was called.
JP #BLUE,@ABS[v2]>2	Jump to #BLUE if the absolute value of variable, v2, is greater than 2
JP #C,v1*v7<=v8*v2	Jump to #C if the value of v1 times v7 is less than or equal to the value of v8*v2
JP#A	Jump to #A

Example Using JP command:

Move the X motor to absolute position 1000 counts and back to zero ten times. Wait 100 msec between moves.

```
#BEGIN          Begin Program
count=10        Initialize loop counter
#LOOP           Begin loop
PA 1000         Position absolute 1000
BGX             Begin move
AMX             Wait for motion complete
WT 100          Wait 100 msec
PA 0            Position absolute 0
BGX             Begin move
AMX             Wait for motion complete
WT 100          Wait 100 msec
count=count-1   Decrement loop counter
JP #LOOP,count>0  Test for 10 times thru loop
EN              End Program
```

Using If, Else, and Endif Commands

The DMC-30000 provides a structured approach to conditional statements using IF, ELSE and ENDIF commands.

Using the IF and ENDIF Commands

An IF conditional statement is formed by the combination of an IF and ENDIF command. The IF command has as it's arguments one or more conditional statements. If the conditional statement(s) evaluates true, the command interpreter will continue executing commands which follow the IF command. If the conditional statement evaluates false, the controller will ignore commands until the associated ENDIF command is executed OR an ELSE command occurs in the program (see discussion of ELSE command below).

NOTE: An ENDIF command must always be executed for every IF command that has been executed. It is recommended that the user not include jump commands inside IF conditional statements since this causes re-direction of command execution. In this case, the command interpreter may not execute an ENDIF command.

Using the ELSE Command

The ELSE command is an optional part of an IF conditional statement and allows for the execution of command only when the argument of the IF command evaluates False. The ELSE command must occur after an IF command and has no arguments. If the argument of the IF command evaluates false, the controller will skip commands until the ELSE command. If the argument for the IF command evaluates true, the controller will execute the commands between the IF and ELSE command.

Nesting IF Conditional Statements

The DMC-30000 allows for IF conditional statements to be included within other IF conditional statements. This technique is known as 'nesting' and the DMC-30000 allows up to 255 IF conditional statements to be nested. This is a very powerful technique allowing the user to specify a variety of different cases for branching.

Command Format - IF, ELSE and ENDIF

Format:	Description
IF conditional statement(s)	Execute commands proceeding IF command (up to ELSE command) if conditional statement(s) is true, otherwise continue executing at ENDIF command or optional ELSE command.
ELSE	Optional command. Allows for commands to be executed when argument of IF command evaluates not true. Can only be used with IF command.
ENDIF	Command to end IF conditional statement. Program must have an ENDIF command for every IF command.

Example using IF, ELSE and ENDIF:

```
#TEST                          Begin Main Program "TEST"
II,,3                          Enable input interrupts on input 1 and input
                            2
MG "WAITING FOR INPUT 1,       Output message
INPUT 2"
#LOOP                          Label to be used for endless loop
JP #LOOP                       Endless loop
EN                             End of main program
#ININT                         Input Interrupt Subroutine
IF (@IN[1]=0)                  IF conditional statement based on input 1
IF (@IN[2]=0)                  2nd IF conditional statement executed if 1st IF
                            conditional true
MG "INPUT 1 AND INPUT 2        Message to be executed if 2nd IF conditional
ARE ACTIVE"                 is true
ELSE                           ELSE command for 2nd IF conditional statement
```

```
     MG "ONLY INPUT 1 IS ACTIVE      Message to be executed if 2nd IF conditional is false
     ENDIF                           End of 2nd conditional statement
     ELSE                            ELSE command for 1st IF conditional statement
     MG"ONLY INPUT 2 IS ACTIVE"      Message to be executed if 1st IF conditional statement is false
     ENDIF                           End of 1st conditional statement
     #WAIT                           Label to be used for a loop
     JP#WAIT,(@IN[1]=0) | (@IN[2]=0) Loop until both input 1 and input 2 are not active
     RI0                             End Input Interrupt Routine without restoring trippoints
```

Subroutines

A subroutine is a group of instructions beginning with a label and ending with an end command (EN). Subroutines are called from the main program with the jump subroutine instruction JS, followed by a label or line number, and conditional statement. Up to 8 subroutines can be nested. After the subroutine is executed, the program sequencer returns to the program location where the subroutine was called unless the subroutine stack is manipulated as described in the following section.

Example:

An example of a subroutine to draw a square 500 counts per side is given below. The square is drawn at vector position 1000,1000.

```
     #M                  Begin Main Program
     CB1                 Clear Output Bit 1 (pick up pen)
     VP 1000,1000;LE;BGS Define vector position; move pen
     AMS                 Wait for after motion trippoint
     SB1                 Set Output Bit 1 (put down pen)
     JS #Square;CB1      Jump to square subroutine
     EN                  End Main Program
     #Square             Square subroutine
     v1=500;JS #L        Define length of side
     v1=-v1;JS #L        Switch direction
     EN                  End subroutine
     #L;PR v1,v1;BGX     Define X,Y; Begin X
     AMX;BGY;AMY         After motion on X, Begin Y
     EN                  End subroutine
```

Stack Manipulation

It is possible to manipulate the subroutine stack by using the ZS command. Every time a JS instruction, interrupt or automatic routine (such as #POSERR or #LIMSWI) is executed, the subroutine stack is incremented by 1. Normally the stack is restored with an EN instruction. Occasionally it is desirable not to return back to the program line where

the subroutine or interrupt was called. The ZS1 command clears 1 level of the stack. This allows the program sequencer to continue to the next line. The ZS0 command resets the stack to its initial value. For example, if a limit occurs and the #LIMSWI routine is executed, it is often desirable to restart the program sequence instead of returning to the location where the limit occurred. To do this, give a ZS command at the end of the #LIMSWI routine.

Auto-Start Routine

The DMC-30000 has a special label for automatic program execution. A program which has been saved into the controller's non-volatile memory can be automatically executed upon power up or reset by beginning the program with the label #AUTO. The program must be saved into non-volatile memory using the command, BP.

Automatic Subroutines for Monitoring Conditions

Often it is desirable to monitor certain conditions continuously without tying up the host or DMC-30000 program sequences. The controller can monitor several important conditions in the background. These conditions include checking for the occurrence of a limit switch, a defined input, position error, or a command error. Automatic monitoring is enabled by inserting a special, predefined label in the applications program. The pre-defined labels are:

SUBROUTINE	DESCRIPTION
#LIMSWI	Limit switch on any axis goes low
#ININT	Input specified by II goes low
#POSERR	Position error exceeds limit specified by ER
#MCTIME	Motion Complete timeout occurred. Timeout period set by TW command
#CMDERR	Bad command given
#AUTO	Automatically executes on power up
#AUTOERR	Automatically executes when a checksum is encountered during #AUTO start-up. Check error condition with _RS. bit 0 for variable checksum error bit 1 for parameter checksum error bit 2 for program checksum error bit 3 for master reset error (there should be no program)
#AMPERR	Error from internal Galil amplifier

For example, the #POSERR subroutine will automatically be executed when any axis exceeds its position error limit. The commands in the #POSERR subroutine could decode which axis is in error and take the appropriate action. In another example, the #ININT label could be used to designate an input interrupt subroutine. When the specified input occurs, the program will be executed automatically.

NOTE: An application program must be running for #CMDERR to function.

Example - Limit Switch:

This program prints a message upon the occurrence of a limit switch. Note, for the #LIMSWI routine to function, the DMC-30000 must be executing an applications program from memory. This can be a very simple program that does nothing but loop on a statement, such as #LOOP;JP #LOOP;EN. Motion commands, such as JG 5000 can still be sent from the PC even while the "dummy" applications program is being executed.

```
#LOOP                   Dummy Program
JP #LOOP;EN             Jump to Loop
#LIMSWI                 Limit Switch Label
```

```
      MG "LIMIT                 Print Message
OCCURRED"
      RE                        Return to main program
                                Download Program
      :XQ #LOOP                 Execute Dummy Program
      :JG 5000                  Jog
      :BGX                      Begin Motion
```

Now, when a forward limit switch occurs on the X axis, the #LIMSWI subroutine will be executed.

Notes regarding the #LIMSWI Routine:

1) The RE command is used to return from the #LIMSWI subroutine.
2) The #LIMSWI subroutine will be re-executed if the limit switch remains active.

The #LIMSWI routine is only executed when the motor is being commanded to move.

Example - Position Error

```
      #LOOP                     Dummy Program
      JP #LOOP;EN               Loop
      #POSERR                   Position Error Routine
      V1=_TEX                   Read Position Error
      MG "EXCESS POSITION       Print Message
ERROR"
      MG "ERROR=",V1=           Print Error
      RE                        Return from Error
                                Download program
      :XQ #LOOP                 Execute Dummy Program
      :JG 100000                Jog at High Speed
      :BGX                      Begin Motion
```

Example - Input Interrupt

```
      #A                        Label
      II1                       Input Interrupt on 1
      JG 30000                  Jog
      BGX                       Begin Motion
      #LOOP;JP#LOOP;EN          Loop
      #ININT                    Input Interrupt
      STX;AM                    Stop Motion
      #TEST;JP #TEST,           Test for Input 1 still low
@IN[1]=0
      JG 30000                  Restore Jog
      BGX                       Begin motion
      RI0                       Return from interrupt routine to Main
                          Program and do not re-enable trippoints
```

Example - Motion Complete Timeout

```
#BEGIN              Begin main program
TW 1000             Set the time out to 1000 ms
PA 10000            Position Absolute command
BGX                 Begin motion
MCX                 Motion Complete trip point
EN                  End main program
#MCTIME             Motion Complete Subroutine
MG "X fell          Send out a message
short"
EN                  End subroutine
```

This simple program will issue the message "X fell short" if the X axis does not reach the commanded position within 1 second of the end of the profiled move.

Example - Command Error

```
#BEGIN                  Begin main program
speed = 2000            Set variable for speed
JG speed;BGX;           Begin motion
#LOOP
JG speed;WT100          Update Jog speed based upon speed
                        variable
JP #LOOP
EN                      End main program
#CMDERR                 Command error utility
JP#DONE,_ED<>2          Check if error on line 2
JP#DONE,_TC<>6          Check if out of range
MG "SPEED TOO           Send message
HIGH"
MG "TRY AGAIN"          Send message
ZS1                     Adjust stack
JP #BEGIN               Return to main program
#DONE                   End program if other error
ZS0                     Zero stack
EN                      End program
```

The above program prompts the operator to enter a jog speed. If the operator enters a number out of range (greater than 8 million), the #CMDERR routine will be executed prompting the operator to enter a new number.

In multitasking applications, there is an alternate method for handling command errors from different threads. Using the XQ command along with the special operands described below allows the controller to either skip or retry invalid commands.

OPERAND	FUNCTION
_ED1	Returns the number of the thread that generated an error
_ED2	Retry failed command (operand contains the location of the failed command)
_ED3	Skip failed command (operand contains the location of the command after the failed

	command)

The operands are used with the XQ command in the following format:

```
XQ _ED2 (or _ED3),_ED1,1
```

Where the ",1" at the end of the command line indicates a restart; therefore, the existing program stack will not be removed when the above format executes.

The following example shows an error correction routine which uses the operands.

Example - Command Error w/Multitasking

```
#A                      Begin thread 0 (continuous loop)
JP#A
EN                      End of thread 0


#B                      Begin thread 1

N=-1                    Create new variable

KP N                    Set KP to value of N, an invalid
                    value
TY                      Issue invalid command

EN                      End of thread 1


#CMDERR                 Begin command error subroutine

IF _TC=6                If error is out of range (KP -1)

N=1                     Set N to a valid number

XQ _ED2,_ED1,1          Retry KP N command

ENDIF
IF _TC=1                If error is invalid command (TY)

XQ _ED3,_ED1,1          Skip invalid command

ENDIF
EN                      End of command error routine
```

Example - Communication Interrupt

A DMC-30000 is used to move the axis back and forth from 0 to 10000. This motion can be paused, resumed and stopped via input from an RS-232 device.

```
#BEGIN                          Label for beginning of program
```

```
CI 2                          Setup communication interrupt for
                            auxiliary serial port
MG {P2}"Type 0 to stop        Message out of auxiliary port
motion"
MG {P2}"Type 1 to pause       Message out of auxiliary port
motion"
MG {P2}"Type 2 to resume      Message out of auxiliary port
motion"
rate=2000                     Variable to remember speed
SPA=rate                      Set speed of A axis motion
#LOOP                         Label for Loop
PAA=10000                     Move to absolute position 10000
BGA                           Begin Motion on A axis
AMA                           Wait for motion to be complete
PAA=0                         Move to absolute position 0
BGA                           Begin Motion on A axis
AMA                           Wait for motion to be complete
JP #LOOP                      Continually loop to make back and forth
                            motion
EN                            End main program
#COMINT                        Interrupt Routine
JP #STOP,P1CH="0"              Check for S (stop motion)
JP #PAUSE,P1CH="1"             Check for P (pause motion)
JP #RESUME,P1CH="2"            Check for R (resume motion)
EN1,1                          Do nothing
#STOP                          Routine for stopping motion
STA;ZS;EN                     Stop motion on A axis; Zero program
                             stack; End Program
#PAUSE                        Routine for pausing motion
rate=_SPA                     Save current speed setting of A axis
                             motion
SPA=0                         Set speed of A axis to zero (allows for
                             pause)
EN1,1                         Re-enable trip-point and communication
                             interrupt
#RESUME                       Routine for resuming motion
SPA=rate                      Set speed on A axis to original speed
EN1,1                         Re-enable trip-point and communication
                             interrupt
```

For additional information, see section on Using Communication Interrupt.

Example – Ethernet Communication Error

This simple program executes in the DMC-30000 and indicates (via the serial port) when a communication handle fails. By monitoring the serial port, the user can re-establish communication if needed.

```
#LOOP                         Simple program loop
```

```
JP#LOOP
EN
#TCPERR                    Ethernet communication error
                         auto routine
MG {P1}_IA4                Send message to serial port
                         indicating which handle did not
                         receive proper acknowledgment.
RE
```

Example – Amplifier Error

The program below will execute upon the detection of an error from an internal Galil Amplifier. The bits in TA1 will be set for all axes that have an invalid hall state even if BR1 is set for those axes, this is handled with the mask variable shown in the code below.

```
#AMPERR
REM mask out if in brushed mode for _TA1
mask=@COM[_BRA]
mask=((_TA1&mask)&$0000FFFF)
REM amplifier error status
MG"A-ER TA0",_TA0
MG"A-ER TA1",mask
MG"A-ER TA2",_TA2
MG"A-ER TA3",_TA3
WT5000
REM the sum of the amperr bits should be 0 with no amplifier error
er=_TA0+mask+_TA2+_TA3
JP#AMPERR,er0
REM Notify user amperr has cleared
MG"AMPERR RESOLVED"
WT3000
RE
```

JS Subroutine Stack Variables (^a, ^b, ^c, ^d, ^e, ^f, ^g, ^h)

There are 8 variables that may be passed on the subroutine stack when using the JS command. Passing values on the stack is advanced DMC programming, and is recommended for experienced DMC programmers familiar with the concept of passing arguments by value and by reference.

Notes:

1. Passing parameters has no type checking, so it is important to exercise good programming style when passing parameters. See examples below for recommended syntax.
2. Do not use spaces in expressions containing ^.
3. Global variables MUST be assigned prior to any use in subroutines where variables are passed by reference.
4. Please refer to the JS command in the controller's command reference for further important information.

Example: A Simple Adding Function

```
#Add
JS#SUM(1,2,3,4,5,6,7,8)              ;' call subroutine, pass values
MG_JS                                ;' print return value
EN
```

```
'
#SUM                                ;NO(^a,^b,^c,^d,^e,^f,^g,^h) syntax note for use
EN,,(^a+^b+^c+^d+^e+^f+^g+^h)       ;' return sum

:Executed program from program1.dmc
36.0000
```

Example: Variable, and an Important Note about Creating Global Variables

```
#Var
value=5                             ;'a value to be passed by reference
global=8                            ;'a global variable
JS#SUM(&value,1,2,3,4,5,6,7)        ;'note first arg passed by reference
MG value                            ;'message out value after subroutine.
MG _JS                              ;'message out returned value
EN
'
#SUM                                ;NO(* ^a,^b,^c,^d,^e,^f,^g)
^a=^b+^c+^d+^e+^f+^g+^h+global
EN,,^a
'notes:
'do not use spaces when working with ^
'If using global variables, they MUST be created before the subroutine is run

Executed program from program2.dmc
36.0000
36.0000
```

Example: Working with Arrays

```
#Array
DM speeds[8]
DM other[256]
JS#zeroAry("speeds",0)              ;'zero out all buckets in speeds[]
JS#zeroAry("other",0)               ;'zero out all buckers in other[]
EN
'
#zeroAry                            ;NO(array ^a, ^b) zeros array starting at index ^b
^a[^b]=0
^b=^b+1
JP#zeroAry,(^b<^a[-1])              ;'[-1] returns the length of an array
EN
```

Example: Abstracting Axes

```
#Axes
JS#runMove(0,10000,1000,100000,100000)
MG "Position:",_JS
EN
'
#runMove            ;NO(axis ^a, PR ^b, SP ^c, AC ^d, DC ^e) Profile movement for axis
~a=^a               ;'~a is global, so use carefully in subroutines
                     'try one variable axis a-h for each thread A-H

PR~a=^b
SP~a=^c
AC~a=^d
```

```
DC~a=^e
BG~a
MC~a
EN,,_TP~a
```

Example: Local Scope

```
#Local
JS#POWER(2,2)
MG_JS
JS#POWER(2,16)
MG_JS
JS#POWER(2,-8)
MG_JS
'
#POWER          ;NO(base ^a,exponent^b) Returns base^exponent power. +/- integer only
^c=1            ;'unpassed variable space (^c-^h) can be used as local scope variables
IF ^b=0         ;'special case, exponent = 0
 EN,,1
ENDIF
IF ^b<0         ;'special case, exponent < 0, invert result
 ^d=1
 ^b=@ABS[^b]
ELSE
 ^d=0
ENDIF
#PWRHLPR
^c=^c*^a
^b=^b-1
JP#PWRHLPR,^b>0
IF ^d=1         ;'if inversion required
 ^c=(1/^c)
ENDIF
EN,,^c

Executed program from program1.dmc
4.0000
65536.0000
0.0039
```

General Program Flow and Timing information

This section will discuss general programming flow and timing information for Galil programming.

REM vs. NO or ' comments

There are 2 ways to add comments to a .dmc program. REM statements or NO/ ' comments. The main difference between the 2 is that REM statements are stripped from the program upon download to the controller and NO or ' comments are left in the program. In most instances the reason for using REM statements instead of NO or ' is to save program memory. The other benefit to using REM commands comes when command execution of a loop, thread or any section of code is critical. Although they do not take much time, NO and ' comments still take time to process. So when command execution time is critical, REM statements should be used. The 2 examples below demonstrate the difference in command execution of a loop containing comments.

Note: Actual processing time will vary depending upon number of axes, communication activity, number of threads currently executing etc.

```
#a
i=0;'initialize a counter
t= TIME;' set an initial time reference
#loop
NO this comment takes time to process
'this comment takes time to process
i=i+1;'this comment takes time `
to process
JP#loop,i<1000
MG TIME-t;'display number of samples`
 from initial time reference
EN
```

When executed on a DMC-30012, the output from the above program returned a 158, which indicates that it took 158 samples (TM 1000) to process the commands from 't=TIME' to 'MG TIME-t'. This is about 154ms +/-2ms.

Now when the comments inside of the #loop routine are changed into REM statements (a REM statement must always start on a new line), the processing is greatly reduced.

When executed on the same DMC-30012, the output from the program shown below returned a 84, which indicates that it took 84 samples to process the commands from 't=TIME' to 'MG TIME-t'. This is about 82ms +/-2ms, and about 50% faster than when the comments where downloaded to the controller.

```
#a
i=0;'initialize a counter
t= TIME;' set an initial time reference
#loop
REM this comment is removed upon download and takes no time to
process
REM this comment is removed upon download and takes no time to
process
i=i+1
REM this comment is removed upon download and takes no time to
process
JP#loop,i<1000
MG TIME-t;'display number of samples`
 from initial time reference
EN
```

WT vs AT and coding deterministic loops

The main difference between WT and AT is that WT will hold up execution of the next command for the specified time from the execution of the WT command, AT will hold up execution of the next command for the specified time from the last time reference set with the AT command.

```
#A
AT0;'set initial AT time reference
WT 1000,1;'wait 1000 samples
```

```
t1=TIME
AT 4000,1;'wait 4000 samples from last time reference
t2=TIME-t1
REM in the above scenario, t2 will be ~3000 because AT 4000,1
will have
REM paused program execution from the time reference of AT0
REM since the WT 1000,1 took 1000 samples, there was only 3000
samples left
REM of the "4000" samples for AT 4000,1
MG t,t2;'this should output 1000,3000
EN;'End program
```

Where the functionality of the operation of the AT command is very useful is when it is required to have a deterministic loop operating on the controller. These instances range from writing PLC-type scan threads to writing custom control algorithms. The key to having a deterministic loop time is to have a trippoint that will wait a specified time independent of the time it took to execute the loop code. In this definition, the AT command is a perfect fit. The below code is an example of a PLC-type scan thread that runs at a 500ms loop rate. A typical implementation would be to run this code in a separate thread (ex XQ#plcscan,2).

```
REM this code will set output 3 high if
REM inputs 1 and 2 are high, and input 3 is low
REM else output 3 will be low
REM if input 4 is low, output 1 will be high
REM and ouput 3 will be low regardless of the
REM states of inputs 1,2 or 3
#plcscan
AT0;'set initial time reference
#scan
REM mask inputs 1-4
ti=_TI0&$F
REM variables for bit 1 and bit 3
b1=0;b3=0
REM if input 4 is high set bit 1 and clear bit 3
REM ti&8 - gets 4th bit, if 4th bit is high result = 8
IF ti&8=8;b1=1;ELSE
REM ti&7 get lower 3 bits, if 011 then result = 3
IF ti&7=3;b3=1;ENDIF;ENDIF
REM set output bits 1 and 3 accordingly
REM set outputs at the end for a PLC scan
OB1,b1;OB3,b3
REM wait 500ms (for 500 samples use AT-500,1)
REM the '-' will reset the time reference
AT-500
JP#scan
```

Mathematical and Functional Expressions

Mathematical Operators

For manipulation of data, the DMC-30000 provides the use of the following mathematical operators:

Operator	Function
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus
&	Logical And (Bit-wise)
|	Logical Or (On some computers, a solid vertical line appears as a broken line)
()	Parenthesis

Mathematical operations are executed from left to right. Calculations within parentheses have precedence.

Examples:

```
speed = 7.5*V1/2          The variable, speed, is equal to 7.5 multiplied by V1 and divided by 2
count = count+2           The variable, count, is equal to the current value plus 2.
result =_TPX-(@COS[45]*40) Puts the position of X - 28.28 in result.  40 * cosine of 45° is 28.28
temp = @IN[1]&@IN[2]      temp is equal to 1 only if Input 1 and Input 2 are high
```

Mathematical Operation Precision and Range

The controller stores non-integers in a fixed point representation (not floating point). Numbers are stored as 4 bytes of integer and 2 bytes of fraction within the range of +/- 2,147,483,647.9999. The smallest number representable (and thus the precision) is 1/65536 or approximately 0.000015.

Example:

Using basic mathematics it is known that 1.4*(80,000) = 112,000. However, using a basic terminal, a DMC controller would calculate the following:

```
:var= 1.4*80000;'       Storing the result of 1.4*80000 in var
:MG var;'               Prints variable "var" to screen
111999.5117
:
```

The reason for this error relies in the precision of the controller. 1.4 must be stored to the nearest multiple of 1/65536, which is 91750/65536 = 1.3999. Thus, (91750/65536)*80000 = 111999.5117 and reveals the source of the error.

By ignoring decimals and multiplying by integers first (since they carry no error), and then adding the decimal back in by dividing by a factor of 10 will allow the user to avoid any errors caused by the limitations of precision of the controller. Continuing from the example above:

```
:var= 14*80000;'        Ignore decimals
:MG var;'               Print result
1120000.0000
:var= var/10;'          Divide by 10 to add in decimal
:MG var;'               Print correct result
112000.0000
:
```

Bit-Wise Operators

The mathematical operators & and | are bit-wise operators. The operator, &, is a Logical And. The operator, |, is a Logical Or. These operators allow for bit-wise operations on any valid DMC-30000 numeric operand, including variables, array elements, numeric values, functions, keywords, and arithmetic expressions. The bit-wise operators may also be used with strings. This is useful for separating characters from an input string. When using the input command for string input, the input variable will hold up to 6 characters. These characters are combined into a single value which is represented as 32 bits of integer and 16 bits of fraction. Each ASCII character is represented as one byte (8 bits), therefore the input variable can hold up to six characters. The first character of the string will be placed in the top byte of the variable and the last character will be placed in the lowest significant byte of the fraction. The characters can be individually separated by using bit-wise operations as illustrated in the following example:

```
      #TEST                    Begin main program
      len="123456"             Set len to a string value
      Flen=@FRAC[len]          Define variable 'Flen' as fractional part of
                               variable 'len'
      Flen=$10000*Flen         Shift Flen by 32 bits (IE - convert fraction,
                               Flen, to integer)
      len1=(Flen&$00FF)        Mask top byte of Flen and set this value to
                               variable 'len1'
      len2=(Flen&$FF00)/       Let variable, 'len2' = top byte of Flen
$100
      len3=len&$000000FF       Let variable, 'len3' = bottom byte of len
      len4=(len&$0000FF0       Let variable, 'len4' = second byte of len
0)/$100
      len5=(len&$00FF000       Let variable, 'len5' = third byte of len
0)/$10000
      len6=(len&$FF00000       Let variable, 'len6' = fourth byte of len
0)/$1000000
      MG len6 {S4}             Display 'len6' as string message of up to 4
                               chars
      MG len5 {S4}             Display 'len5' as string message of up to 4
                               chars
      MG len4 {S4}             Display 'len4' as string message of up to 4
                               chars
      MG len3 {S4}             Display 'len3' as string message of up to 4
                               chars
      MG len2 {S4}             Display 'len2' as string message of up to 4
                               chars
      MG len1 {S4}             Display 'len1' as string message of up to 4
                               chars
      EN
```

This program will accept a string input of up to 6 characters, parse each character, and then display each character. Notice also that the values used for masking are represented in hexadecimal (as denoted by the preceding '$'). For more information, see section Sending Messages.

To illustrate further, if the user types in the string "TESTME" at the input prompt, the controller will respond with the following:

```
              T                        Response from command MG len6
```

```
                              {S4}
     E                             Response from command MG len5
                              {S4}
     S                             Response from command MG len4
                              {S4}
     T                             Response from command MG len3
                              {S4}
     M                             Response from command MG len2
                              {S4}
     E                             Response from command MG len1
                              {S4}
```

Functions

FUNCTION	DESCRIPTION
@SIN[n]	Sine of n (n in degrees, with range of -32768 to 32767 and 16-bit fractional resolution)
@COS[n]	Cosine of n (n in degrees, with range of -32768 to 32767 and 16-bit fractional resolution)
@TAN[n]	Tangent of n (n in degrees, with range of -32768 to 32767 and 16-bit fractional resolution)
@ASIN*[n]	Arc Sine of n, between -90° and +90°. Angle resolution in 1/64000 degrees.
@ACOS*[n]	Arc Cosine of n, between 0 and 180°. Angle resolution in 1/64000 degrees.
@ATAN*[n]	Arc Tangent of n, between -90° and +90°. Angle resolution in 1/64000 degrees
@COM[n]	1's Complement of n
@ABS[n]	Absolute value of n
@FRAC[n]	Fraction portion of n
@INT[n]	Integer portion of n
@RND[n]	Round of n (Rounds up if the fractional part of n is .5 or greater)
@SQR[n]	Square root of n (Accuracy is +/-.004)
@IN[n]	Return digital input at general input n (where n starts at 1)
@OUT[n]	Return digital output at general output n (where n starts at 1)
@AN[n]	Return analog input at general analog in n (where n starts at 1)

*Note that these functions are multi-valued. An application program may be used to find the correct band.

Functions may be combined with mathematical expressions. The order of execution of mathematical expressions is from left to right and can be over-ridden by using parentheses.

Examples:

```
     v1=@ABS[V7]      The variable, v1, is equal to the absolute value of
                      variable v7.
     v2=5*@SIN[pos]   The variable, v2, is equal to five times the sine
                      of the variable, pos.
     v3=@IN[1]        The variable, v3, is equal to the digital value of
                      input 1.
     v4=2*(5+@AN[5])  The variable, v4, is equal to the value of analog
                      input 5 plus 5, then multiplied by 2.
```

Variables

For applications that require a parameter that is variable, the DMC-30000 provides 254 variables. These variables can be numbers or strings. A program can be written in which certain parameters, such as position or speed, are defined as variables. The variables can later be assigned by the operator or determined by program calculations. For example, a cut-to-length application may require that a cut length be variable.

Example:

```
posx=5000          Assigns the value of 5000 to the variable
                posx
PR posx            Assigns variable posx to PR command
JG rpmY*70         Assigns variable rpmY multiplied by 70 to JG
                command.
```

Programmable Variables

The DMC-30000 allows the user to create up to 254 variables. Each variable is defined by a name which can be up to eight characters. The name must start with an alphabetic character; however, numbers are permitted in the rest of the name. Spaces are not permitted. Variable names should not be the same as DMC-30000 instructions. For example, PR is not a good choice for a variable name.

NOTE: It is generally a good idea to use lower-case variable names so there is no confusion between Galil commands and variable names.

Examples of valid and invalid variable names are:

Valid Variable Names

```
posx
pos1
speedZ
```

Invalid Variable Names

```
RealLongName      ; 'Cannot have more than 8 characters
123          ; 'Cannot begin variable name with a number
speed Z            ; 'Cannot have spaces in the name
```

Assigning Values to Variables:

Assigned values can be numbers, internal variables and keywords, functions, controller parameters and strings. The range for numeric variable values is 4 bytes of integer (231) followed by two bytes of fraction (+/- 2,147,483,647.9999).

Numeric values can be assigned to programmable variables using the equal sign.

Any valid DMC-30000 function can be used to assign a value to a variable. For example, v1=@ABS[v2] or v2=@IN[1]. Arithmetic operations are also permitted.

To assign a string value, the string must be in quotations. String variables can contain up to six characters which must be in quotation.

Examples:

```
posX=_TPX          Assigns returned value from TPX command to
                variable posx.
speed=5.75         Assigns value 5.75 to variable speed
input=@IN[2        Assigns logical value of input 2 to variable
]                  input
```

```
v2=v1+v3*v4       Assigns the value of v1 plus v3 times v4 to
                the variable v2.
var=”CAT”         Assign the string, CAT, to var
MG var{S3}        Displays the variable var – (CAT)
```

Assigning Variable Values to Controller Parameters

Variable values may be assigned to controller parameters such as SP or PR.

```
PR v1             Assign v1 to PR command
SP vS*2000        Assign vS*2000 to SP command
```

Displaying the value of variables at the terminal

Variables may be sent to the screen using the format, variable=. For example, v1= , returns the value of the variable v1.

Example - Using Variables for Joystick

The example below reads the voltage of an X-Y joystick and assigns it to variables vX and vY to drive the motors at proportional velocities, where:

10 Volts = 3000 rpm = 200000 c/sec

Speed/Analog input = 200000/10 = 20000

```
      #JOYSTI       Label
K
      JG 0          Set in Jog mode
      BGX           Begin Motion
      AT0           Set AT time reference
      #LOOP         Loop
      vX=@AN[       Read joystick X
1]*20000
      JG vX         Jog at variable vX
      AT-4          Wait 4ms from last time reference, creates a
                deterministic loop time
      JP#LOOP       Repeat
      EN            End
```

Operands

Operands allow motion or status parameters of the DMC-30000 to be incorporated into programmable variables and expressions. Most DMC commands have an equivalent operand - which are designated by adding an underscore (_) prior to the DMC-30000 command. The command reference indicates which commands have an associated operand.

Status commands such as Tell Position return actual values, whereas action commands such as KP or SP return the values in the DMC-30000 registers. The axis designation is required following the command.

Examples of Internal Variables:

```
posX=_TPX         Assigns value from Tell Position X to the
                variable posX.
```

```
     deriv=_KDX*       Assigns value from KDX multiplied by two to
2                 variable, deriv.
     JP                Jump to #LOOP if the position error of X is
#LOOP,_TEX>5      greater than 5
     JP                Jump to #ERROR if the error code equals 1.
#ERROR,_TC=1
```

Operands can be used in an expression and assigned to a programmable variable, but they cannot be assigned a value. For example: _KDX=2 is invalid.

Special Operands (Keywords)

The DMC-30000 provides a few additional operands which give access to internal variables that are not accessible by standard DMC-30000 commands.

Keyword	Function
_BGA	*Returns a 1 if motion on the axis, otherwise returns 0.
_BN	*Returns serial # of the board.
_DA	*Returns the number of arrays available
_DL	*Returns the number of available labels for programming
_DM	*Returns the available array memory
_HMA	*Returns status of Home Switch (equals 0 or 1)
_LFA	Returns status of Forward Limit switch input (equals 0 or 1)
_LRA	Returns status of Reverse Limit switch input (equals 0 or 1)
_UL	*Returns the number of available variables
TIME	Free-Running Real Time Clock (off by 2.4% - Resets with power-on). Note: TIME does not use an underscore character (_) as other keywords.

* - These keywords have corresponding commands while the keywords _LF, _LR, and TIME do not have any associated commands. All keywords are listed in the Command Reference.

Examples of Keywords:

```
     v1=       Assign V1 the logical state of the Forward Limit
_LFA      Switch
     v3=       Assign V3 the current value of the time clock
TIME
     v4=       Assign V4 the logical state of the Home input
_HMA
```

Arrays

For storing and collecting numerical data, the DMC-30000 provides array space for 3000 elements. The arrays are one dimensional and up to 6 different arrays may be defined. Each array element has a numeric range of 4 bytes of integer (2^{31}) followed by two bytes of fraction (+/-2,147,483,647.9999).

Arrays can be used to capture real-time data, such as position, torque and analog input values. In the contouring mode, arrays are convenient for holding the points of a position trajectory in a record and playback application.

Defining Arrays

An array is defined with the command DM. The user must specify a name and the number of entries to be held in the array. An array name can contain up to eight characters, starting with an uppercase alphabetic character. The number of entries in the defined array is enclosed in [ ].

Example:

```
DM posx[7]          Defines an array names 'posx' with seven
                    entries
DM                  Defines an array named speed with 100
speed[100]          entries
DA posx[]           Frees array space
```

Assignment of Array Entries

Like variables, each array element can be assigned a value. Assigned values can be numbers or returned values from instructions, functions and keywords.

Array elements are addressed starting at count 0. For example the first element in the 'posx' array (defined with the DM command, DM posx[7]) would be specified as posx[0].

Values are assigned to array entries using the equal sign. Assignments are made one element at a time by specifying the element number with the associated array name.

NOTE: Arrays must be defined using the command, DM, before assigning entry values.

Examples:

```
DM                  Dimension speed Array
speed[10]
speed[0]=76         Assigns the first element of the array,
50.2                'speed' the value 7650.2
speed[0]=           Returns array element value
posx[9]=_TP         Assigns the 10th element of the array 'posx'
X                   the returned value from the tell position
                    command.
con[1]=@COS         Assigns the second element of the array
[pos]*2             'con' the cosine of the variable POS multiplied
                    by 2.
timer[0]=TI         Assigns the first element of the array timer
ME                  the returned value of the TIME keyword.
```

Using a Variable to Address Array Elements

An array element number can also be a variable. This allows array entries to be assigned sequentially using a counter.

Example:

```
#A                  Begin Program
count=0;DM          Initialize counter and define array
pos[10]
#LOOP               Begin loop
WT 10               Wait 10 msec
pos[count]=_        Record position into array element
TPX
pos[count]=         Report position
```

```
      count=count+       Increment counter
1
      JP                 Loop until 10 elements have been stored
#LOOP,count<10
      EN                 End Program
```

The above example records 10 position values at a rate of one value per 10 msec. The values are stored in an array named 'pos'. The variable, 'count', is used to increment the array element counter. The above example can also be executed with the automatic data capture feature described below.

Uploading and Downloading Arrays to On Board Memory

The GalilTools software is recommended for downloading and uploading array data from the controller. The GalilTools Communication library also provides function calls for downloading and uploading array data from the controller to/from a buffer or a file.

Arrays may also be uploaded and downloaded using the QU and QD commands.

QU array[],start,end,delim

QD array[],start,end

where array is an array name such as A[].

start is the first element of array (default=0)

end is the last element of array (default=last element)

delim specifies whether the array data is separated by a comma (delim=1) or a carriage return (delim=0).

The file is terminated using <control>Z, <control>Q, <control>D or \.

Automatic Data Capture into Arrays

The DMC-30000 provides a special feature for automatic capture of data such as position, position error, inputs or torque. This is useful for teaching motion trajectories or observing system performance. Up to six types of data can be captured and stored in six arrays. The capture rate or time interval may be specified. Recording can done as a one time event or as a circular continuous recording.

Command Summary - Automatic Data Capture

Command	Description
RA n[ ],m[ ],o[ ],p[ ]	Selects up to eight arrays for data capture. The arrays must be defined with the DM command.
RD type1,type2,type3,type4	Selects the type of data to be recorded, where type1, type2, type3, and type 4 represent the various types of data (see table below). The order of data type is important and corresponds with the order of n,m,o,p arrays in the RA command.
RC n,m	The RC command begins data collection. Sets data capture time interval where n is an integer between 1 and 8 and designates 2^n msec between data. m is optional and specifies the number of elements to be captured. If m is not defined, the number of elements defaults to the smallest array defined by DM. When m is a negative number, the recording is done continuously in a circular manner. _RD is the recording pointer and indicates the address of the next array element. n=0 stops recording.
RC?	Returns a 0 or 1 where, 0 denotes not recording, 1 specifies recording in progress

Data Types for Recording:

Data type	Description
TIME	Controller time as reported by the TIME command
_AFA	Analog input
_DEA	2nd encoder position (dual encoder)
_NO	Status bits
_OP	Output
_RLA	Latched position
_RPA	Commanded position
_SCA	Stop code
_TEA	Position error
_TI	Inputs
_TPA	Encoder position
_TSA	Switches (only bit 0-4 valid)
_TTA	Torque (reports digital value +/-32544)

NOTE: X may be replaced by Y,Z or W for capturing data on other axes.

Operand Summary - Automatic Data Capture

_RC	Returns a 0 or 1 where, 0 denotes not recording, 1 specifies recording in progress
_RD	Returns address of next array element.

Example - Recording into An Array

During a position move, store the X and Y positions and position error every 2 msec.

```
      #RECORD                    Begin program
      DM                         Define X,Y position arrays
XPOS[300],YPOS[300]
      DM                         Define X,Y error arrays
XERR[300],YERR[300]
      RA                         Select arrays for capture
XPOS[],XERR[],YPOS[],YERR
[]
      RD                         Select data types
_TPX,_TEX,_TPY,_TEY
      PR 10000,20000             Specify move distance
      RC1                        Start recording now, at rate of 2
                           msec
      BG XY                      Begin motion
      #A;JP #A,_RC=1             Loop until done
      MG "DONE"                  Print message
      EN                         End program
      #PLAY                      Play back
      N=0                        Initial Counter
      JP# DONE,N>300             Exit if done
      N=                         Print Counter
```

```
X POS[N]=                Print X position
Y POS[N]=                Print Y position
XERR[N]=                 Print X error
YERR[N]=                 Print Y error
N=N+1                    Increment Counter
#DONE                    Done
EN                       End Program
```

De-allocating Array Space

Array space may be de-allocated using the DA command followed by the array name. DA*[0] deallocates all the arrays.

Input of Data (Numeric and String)

NOTE: The IN command has been removed from the DMC-30000 firmware. Variables should be entered by sending data directly from the host application.

Sending Data from a Host

The DMC-30000 can accept ASCII strings from a host. This is the most common way to send data to the controller such as setting variables to numbers or strings. Any variable can be stored in a string format up to 6 characters by simply specifying defining that variable to the string value with quotes, for example:

```
varS = "STRING"
```

Will assign the variable 'varS' to a string value of "STRING".

To assign a variable a numerical value, the direct number is used, for example:

```
varN = 123456
```

Will assign the variable 'varN' to a number of 123,456.

All variables on the DMC-30000 controller are stored with 6 bytes of integer and 4 bytes of fractional data.

Operator Data Entry Mode

The Operator Data Entry Mode provides for un-buffered data entry through the main RS-232 port. In this mode, the DMC-30000 provides a buffer for receiving characters. This mode may only be used when executing an applications program.

The Operator Data Entry Mode may be specified for Port 2 only. This mode may be exited with the \ or <escape> key.

NOTE: Operator Data Entry Mode cannot be used for high rate data transfer.

To capture and decode characters in the Operator Data Mode, the DMC-30000 provides special the following keywords:

Keyword	Function
P1CH	Contains the last character received
P1ST	Contains the received string
P1NM	Contains the received number

P1CD	Contains the status code: -1 mode disabled 0 nothing received 1 received character, but not <enter> 2 received string, not a number 3 received number

NOTE: The value of P1CD returns to zero after the corresponding string or number is read.

These keywords may be used in an applications program to decode data and they may also be used in conditional statements with logical operators.

Example

Instruction	Interpretation
JP #LOOP,P1CD< >3	Checks to see if status code is 3 (number received)
JP #P,P1CH="V"	Checks if last character received was a V
PR P1NM	Assigns received number to position
JS #XAXIS,P1ST="X"	Checks to see if received string is X

Using Communication Interrupt

The DMC-30000 provides a special interrupt for communication allowing the application program to be interrupted by input from the user. The interrupt is enabled using the CI command. The syntax for the command is CI n:

n = 0	Don't interrupt Port 1
n = 1	Interrupt on <enter> Port 1
n = 2	Interrupt on any character Port 1
n = -1	Clear any characters in buffer

The #COMINT label is used for the communication interrupt. For example, the DMC-30000 can be configured to interrupt on any character received on Port 2. The #COMINT subroutine is entered when a character is received and the subroutine can decode the characters. At the end of the routine the EN command is used. EN,1 will re-enable the interrupt and return to the line of the program where the interrupt was called, EN will just return to the line of the program where it was called without re-enabling the interrupt. As with any automatic subroutine, a program must be running in thread 0 at all times for it to be enabled.

Example

A DMC-30000 is used to jog the axis. The speed of either axis may be changed during motion by specifying the axis letter followed by the new speed value. An S stops motion on both axes.

Instruction	Interpretation
#AUTO	Label for Auto Execute
speedA=10000	Initial A speed
speedB=10000	Initial B speed
CI 2	Set Port 1 for Character Interrupt
JG speedA	Specify jog mode speed
BGX	Begin motion
#PRINT	Routine to print message to terminal

```
      MG{P1}"TO CHANGE          Print message
SPEEDS"
      MG{P1}"TYPE A"
      MG{P1}"TYPE S TO
STOP"
      #JOGLOOP                  Loop to change Jog speeds
      JG speedA                 Set new jog speed
      JP #JOGLOOP
      EN                        End of main program
      #COMINT                   Interrupt routine
      JP #A,P2CH="A"            Check for A
      JP #C,P2CH="S"            Check for S
      ZS1;CI2;JP#JOGLO          Jump if not X,Y,S
OP
      #A;JS#NUM
      speedX=val                New X speed
      ZS1;CI2;JP#PRINT          Jump to Print
      #C;ST;AMX;CI-1            Stop motion on S
      MG{^8}, "THE
END"
      ZS;EN,1                   End-Re-enable interrupt
      #NUM                      Routine for entering new jog
                              speed
      MG                        Prompt for value
"ENTER",P1CH{S},"AXIS
SPEED" {N}
      #NUMLOOP; CI-1            Check for enter
      #NMLP                     Routine to check input from
                              terminal
      JP #NMLP,P1CD<2           Jump to error if string
      JP #ERROR,P1CD=2          Read value
      val=P1NM
      EN                        End subroutine
      #ERROR;CI-1               Error Routine
      MG "INVALID-TRY           Error message
AGAIN"
      JP #NMLP
      EN                        End
```

Output of Data (Numeric and String)

Numerical and string data can be output from the controller using several methods. The message command, MG, can output string and numerical data. Also, the controller can be commanded to return the values of variables and arrays, as well as other information using the interrogation commands (the interrogation commands are described in chapter 5).

Sending Messages

Messages may be sent to the bus using the message command, MG. This command sends specified text and numerical or string data from variables or arrays to the screen.

Text strings are specified in quotes and variable or array data is designated by the name of the variable or array. For example:

```
MG "The Final Value is", result
```

In addition to variables, functions and commands, responses can be used in the message command. For example:

```
MG "Analog input is", @AN[1]
MG "The Position of A is", _TPA
```

Specifying the Port for Messages:

The port can be specified with the specifier, {P1} for the RS-232 port, or {En} for the Ethernet port.

```
MG {P1} "Hello World"          Sends message to RS-232 Port
```

Formatting Messages

String variables can be formatted using the specifier, {Sn} where n is the number of characters, 1 thru 6. For example:

```
MG STR {S3}
```

This statement returns 3 characters of the string variable named STR.

Numeric data may be formatted using the {Fn.m} expression following the completed MG statement. {$n.m} formats data in HEX instead of decimal. The actual numerical value will be formatted with n characters to the left of the decimal and m characters to the right of the decimal. Leading zeros will be used to display specified format.

For example:

```
MG "The Final Value is", result {F5.2}
```

If the value of the variable result is equal to 4.1, this statement returns the following:

The Final Value is 00004.10

If the value of the variable result is equal to 999999.999, the above message statement returns the following:

The Final Value is 99999.99

The message command normally sends a carriage return and line feed following the statement. The carriage return and the line feed may be suppressed by sending {N} at the end of the statement. This is useful when a text string needs to surround a numeric value.

Example:

```
 #A
JG 50000;BGA;ASA
MG "The Speed is", _TVA {F5.0} {N}
MG "counts/sec"
EN
```

When #A is executed, the above example will appear on the screen as:

The Speed is 50000 counts/sec

Using the MG Command to Configure Terminals

The MG command can be used to configure a terminal. Any ASCII character can be sent by using the format {^n} where n is any integer between 1 and 255.

Example:

```
MG {^07} {^255}
```

sends the ASCII characters represented by 7 and 255 to the bus.

Summary of Message Functions

function	description
" "	Surrounds text string
{Fn.m}	Formats numeric values in decimal n digits to the left of the decimal point and m digits to the right
{P1} or {En}	Send message to RS-232 Port or Ethernet Port
{$n.m}	Formats numeric values in hexadecimal
{^n}	Sends ASCII character specified by integer n
{N}	Suppresses carriage return/line feed
{Sn}	Sends the first n characters of a string variable, where n is 1 thru 6.

Displaying Variables and Arrays

Variables and arrays may be sent to the screen using the format, variable= or array[x]=. For example, v1= returns the value of v1.

Example - Printing a Variable and an Array element

Instruction	Interpretation
#DISPLAY	Label
DM posA[7]	Define Array posA with 7 entries
PR 1000	Position Command
BGX	Begin
AMX	After Motion
v1=_TPA	Assign Variable v1
posA[1]=_TPA	Assign the first entry
v1=	Print v1

Interrogation Commands

The DMC-30000 has a set of commands that directly interrogate the controller. When these command are entered, the requested data is returned in decimal format on the next line followed by a carriage return and line feed. The format of the returned data can be changed using the Position Format (PF), and Leading Zeros (LZ) command. For a complete description of interrogation commands, see Chapter 5.

Using the PF Command to Format Response from Interrogation Commands

The command, PF, can change format of the values returned by theses interrogation commands:

BL ?	LE ?
DE ?	PA ?
DP ?	PR ?
EM ?	TN ?
FL ?	VE ?
IP ?	TE

TP

The numeric values may be formatted in decimal or hexadecimal with a specified number of digits to the right and left of the decimal point using the PF command.

Position Format is specified by:

PF m.n

where m is the number of digits to the left of the decimal point (0 thru 10) and n is the number of digits to the right of the decimal point (0 thru 4) A negative sign for m specifies hexadecimal format.

Hex values are returned preceded by a $ and in 2's complement. Hex values should be input as signed 2's complement, where negative numbers have a negative sign. The default format is PF 10.0.

If the number of decimal places specified by PF is less than the actual value, a nine appears in all the decimal places.

Example

Instruction	**Interpretation**
:DP21	Define position
:TPA	Tell position
0000000021	Default format
:PF4	Change format to 4 places
:TPA	Tell position
0021	New format
:PF-4	Change to hexadecimal format
:TPA	Tell Position
$0015	Hexadecimal value
:PF2	Format 2 places
:TPA	Tell Position
99	Returns 99 if position greater than 99

Adding Leading Zeros from Response to Interrogation Commands

The leading zeros on data returned as a response to interrogation commands can be added by the use of the command, LZ. The LZ command is set to a default of 1.

LZ0	Disables the LZ function
TP	Tell Position Interrogation Command
-0000000009, 0000000005	Response (With Leading Zeros)
LZ1	Enables the LZ function
TP	Tell Position Interrogation Command
-9, 5	Response (Without Leading Zeros)

Local Formatting of Response of Interrogation Commands

The response of interrogation commands may be formatted locally. To format locally, use the command, {Fn.m} or {$n.m} on the same line as the interrogation command. The symbol F specifies that the response should be returned

in decimal format and $ specifies hexadecimal. n is the number of digits to the left of the decimal, and m is the number of digits to the right of the decimal.

```
TP {F2.2}                          Tell Position in decimal format 2.2
-05.00, 05.00, 00.00, 07.00        Response from Interrogation Command
TP {$4.2}                          Tell Position in hexadecimal format 4.2
FFFB.00,$0005.00,$0000.00,$0007.00 Response from Interrogation Command
```

Formatting Variables and Array Elements

The Variable Format (VF) command is used to format variables and array elements. The VF command is specified by:

```
VF m.n
```

where m is the number of digits to the left of the decimal point (0 thru 10) and n is the number of digits to the right of the decimal point (0 thru 4).

A negative sign for m specifies hexadecimal format. The default format for VF is VF 10.4

Hex values are returned preceded by a $ and in 2's complement.

Instruction	Interpretation
v1=10	Assign v1
v1=	Return v1
:0000000010.0000	Response - Default format
VF2.2	Change format
v1=	Return v1
:10.00	Response - New format
VF-2.2	Specify hex format
v1=	Return v1
$0A.00	Response - Hex value
VF1	Change format
v1=	Return v1
:9	Response - Overflow

Local Formatting of Variables

PF and VF commands are global format commands that affect the format of all relevant returned values and variables. Variables may also be formatted locally. To format locally, use the command, {Fn.m} or {$n.m} following the variable name and the '=' symbol. F specifies decimal and $ specifies hexadecimal. n is the number of digits to the left of the decimal, and m is the number of digits to the right of the decimal.

Instruction	Interpretation
v1=10	Assign v1
v1=	Return v1
:0000000010.0000	Default Format
v1={F4.2}	Specify local format

```
     :0010.00                New format
v1={$4.2}                    Specify hex format
     :$000A.00               Hex value
v1="ALPHA"                   Assign string "ALPHA" to v1
v1={S4}                      Specify string format first
                          4 characters
     :ALPH
```

The local format is also used with the MG command.

Converting to User Units

Variables and arithmetic operations make it easy to input data in desired user units such as inches or RPM.

The DMC-30000 position parameters such as PR, PA and VP have units of quadrature counts. Speed parameters such as SP, JG and VS have units of counts/sec. Acceleration parameters such as AC, DC, VA and VD have units of counts/sec2. The controller interprets time in milliseconds.

All input parameters must be converted into these units. For example, an operator can be prompted to input a number in revolutions. A program could be used such that the input number is converted into counts by multiplying it by the number of counts/revolution.

Instruction	**Interpretation**
`#RUN`	Label
`MG "ENTER # OF REVOLUTIONS";n1=-1`	Prompt for revs
`#rev;JP#rev,n1=-1`	Wait until user enters new value for n1
`PR n1*2000`	Convert to counts
`MG "ENTER SPEED IN RPM";s1=-1`	Prompt for RPMs
`#spd;JP#spd,s1=-1`	Wait for user to enter new value for s1
`SP s1*2000/60`	Convert to counts/sec
`MG "ENTER ACCEL IN RAD/SEC2";a1=-1`	Prompt for ACCEL
`#acc;JP#acc,a1=-1`	Wait for user to enter new value for a1
`AC a1*2000/(2*3.14)`	Convert to counts/sec2
`BG`	Begin motion
`EN`	End program

Hardware I/O

Digital Outputs

The DMC-30000 has 4-bit uncommitted digital outputs output port. Each bit may be set and cleared with the software instructions SB (Set Bit) and CB (Clear Bit), or OB (define output bit).

Example- Set Bit and Clear Bit

Instruction	Interpretation
SB3	Sets bit 3 of output port
CB4	Clears bit 4 of output port

Example- Output Bit

The Output Bit (OB) instruction is useful for setting or clearing outputs depending on the value of a variable, array, input or expression. Any non-zero value results in a set bit.

Instruction	Interpretation
OB1, POS	Set Output 1 if the variable POS is non-zero. Clear Output 1 if POS equals 0.
OB 2, @IN [1]	Set Output 2 if Input 1 is high. If Input 1 is low, clear Output 2.
OB 3, @IN [1]&@IN [2]	Set Output 3 only if Input 1 and Input 2 are high.
OB 4, COUNT [1]	Set Output 4 if element 1 in the array COUNT is non-zero.

The output port can be set by specifying an 16-bit word using the instruction OP (Output Port). This instruction allows a single command to define the state of the entire 16-bit output port, where bit 0 is output 1, bit1 is output2 and so on. A 1 designates that the output is on.

Example- Output Port

Instruction	Interpretation
OP6	Sets outputs 2 and 3 of output port to high. All other bits are 0. $(2^1 + 2^2 = 6)$
OP0	Clears all bits of output port to zero
OP 15	Sets all bits of output port to one. $(2^0 + 2^1 + 2^2 + 2^3)$

The output port is useful for setting relays or controlling external switches and events during a motion sequence.

Example - Turn on output after move

Instruction	Interpretation
#OUTPUT	Label
PR 2000	Position Command
BG	Begin
AM	After move
SB1	Set Output 1
WT 1000	Wait 1000 msec
CB1	Clear Output 1
EN	End

Digital Inputs

The general digital inputs for are accessed by using the @IN[n] function or the TI command. The @IN[n] function returns the logic level of the specified input, n, where n is a number 1 through 8.

Example - Using Inputs to control program flow

Instruction	Interpretation
JP #A,@IN[1]=0	Jump to A if input 1 is low
JP #B,@IN[2]=1	Jump to B if input 2 is high
AI 7	Wait until input 7 is high
AI -6	Wait until input 6 is low

Example - Start Motion on Switch

Motor A must turn at 4000 counts/sec when the user flips a panel switch to on. When panel switch is turned to off position, motor A must stop turning.

Solution: Connect panel switch to input 1 of DMC-30000. High on input 1 means switch is in on position.

Instruction	Interpretation
#S;JG 4000	Set speed
AI 1;BGA	Begin after input 1 goes high
AI -1;STA	Stop after input 1 goes low
AMA;JP #S	After motion, repeat
EN	

The Auxiliary Encoder Inputs

The auxiliary encoder inputs can be used for general use. The controller has one auxiliary encoder which consists of two inputs, channel A and channel B. The auxiliary encoder input is mapped to the inputs 81 and 82.

Tthe auxiliary encoder is a differential line receiver and can accept voltage levels between +/- 12 volts. The inputs have been configured to accept TTL level signals. To connect TTL signals, simply connect the signal to the + input and leave the - input disconnected. For other signal levels, the - input should be connected to a voltage that is ½ of the full voltage range (for example, connect the - input to 5 volts if the signal is a 0 - 12 volt logic).

NOTE: The auxiliary encoder inputs are not available for any axis that is configured for stepper motor.

Input Interrupt Function

The DMC-30000 provides an input interrupt function which causes the program to automatically execute the instructions following the #ININT label. This function is enabled using the II m,n,o command. The m specifies the beginning input and n specifies the final input in the range. The parameter o is an interrupt mask. If m and n are unused, o contains a number with the mask. For example, II,,5 enables inputs 1 and 3.

A low input on any of the specified inputs will cause automatic execution of the #ININT subroutine. The Return from Interrupt (RI) command is used to return from this subroutine to the place in the program where the interrupt had occurred.

Important: Use the RI command (not EN) to return from the #ININT subroutine.

Example - Input Interrupt

Instruction	Interpretation
#A	Label #A

```
      II 1                    Enable input 1 for interrupt
                            function
      JG 30000,-20000         Set speeds on A and B axes
      BG AB                   Begin motion on A and B axes
      #B                      Label #B
      TP AB                   Report A and B axes positions
      WT 1000                 Wait 1000 milliseconds
      JP #B                   Jump to #B
      EN                      End of program
      #ININT                  Interrupt subroutine
      MG "Interrupt has       Displays the message
occurred"
      ST AB                   Stops motion on A and B axes
      #LOOP;JP                Loop until Interrupt cleared
#LOOP,@IN[1]=0
      JG 15000,10000          Specify new speeds
      WT 300                  Wait 300 milliseconds
      BG AB                   Begin motion on A and B axes
      RI                      Return from Interrupt subroutine
```

Jumping back to main program with #ININT

To jump back to the main program using the JP command, the RI command must be issued in a subroutine and then the ZS command must be issued prior to the JP command. See Application Note # 2418 for more information.

http://www.galilmc.com/support/appnotes/optima/note2418.pdf

Analog Inputs

The DMC-30000 provides two analog inputs. The value of these inputs in volts may be read using the @AN[n] function where n is the analog input 1 through 2. The resolution of the Analog-to-Digital conversion is 12 bits. Analog inputs are useful for reading special sensors such as temperature, tension or pressure.

The following examples show programs which cause the motor to follow an analog signal. The first example is a point-to-point move. The second example shows a continuous move.

Example - Position Follower (Point-to-Point)

Objective - The motor must follow an analog signal. When the analog signal varies by 5V, motor must move 5000 counts.

Method: Read the analog input and command A to move to that point.

```
Instruction             Interpretation
      #POINTS           Label
      SP 7000           Speed
      AC 80000;DC       Acceleration
80000
      #LOOP
      VP=@AN[1]*1       Read and analog input, compute position
000
```

```
PA VP              Command position
BGA                Start motion
AMA                After completion
JP #LOOP           Repeat
EN                 End
```

Example - Position Follower (Continuous Move)

Method: Read the analog input, compute the commanded position and the position error. Command the motor to run at a speed in proportions to the position error.

```
Instruction          Interpretation
     #CONT           Label
     AC              Acceleration rate
80000;DC 80000
     JG 0            Start job mode
     BGX             Start motion
     #LOOP
     vp=@AN[1]*      Compute desired position
1000
     ve=vp-_TPA      Find position error
     vel=ve*20       Compute velocity
     JG vel          Change velocity
     JP #LOOP        Change velocity
     EN              End
```

Example – Low Pass Digital Filter for the Analog inputs

Because the analog inputs on the Galil controller can be used to close a position loop, they have a very high bandwidth and will therefor read noise that comes in on the analog input. Often when an analog input is used in a motion control system, but not for closed loop control, the higher bandwidth is not required. In this case a simple digital filter may be applied to the analog input, and the output of the filter can be used for in the motion control application. This example shows how to apply a simple single pole low-pass digital filter to an analog input. This code is commonly run in a separate thread (XQ#filt,1 – example of executing in thread 1).

```
     #filt
     REM an1 = filtered output.  Use this instead of @AN[1]
     an1=@AN[1];'set initial value
     REM k1+k2=1 this condition must be met
     REM use division of m/2^n for elimination of round off
     REM increase k1 = less filtering
     REM increase k2 = more filtering
     k1=32/64;k2=32/64
     AT0;'set initial time reference
     #loop
     REM calculate filtered output and then way 2 samples from
last
     REM time reference (last AT-2,1 or AT0)
     an1=(k1*@AN[1])+(k2*an1);AT-2,1
     JP#loop
```

Example Applications

Wire Cutter

An operator activates a start switch. This causes a motor to advance the wire a distance of 10". When the motion stops, the controller generates an output signal which activates the cutter. Allowing 100 ms for the cutting completes the cycle.

Suppose that the motor drives the wire by a roller with a 2" diameter. Also assume that the encoder resolution is 1000 lines per revolution. Since the circumference of the roller equals 2π inches, and it corresponds to 4000 quadrature, one inch of travel equals:

$$4000/2\pi = 637 \text{ count/inch}$$

This implies that a distance of 10 inches equals 6370 counts, and a slew speed of 5 inches per second, for example, equals 3185 count/sec.

The input signal may be applied to I1, for example, and the output signal is chosen as output 1. The motor velocity profile and the related input and output signals are shown in Figure 7.1.

The program starts at a state that we define as #A. Here the controller waits for the input pulse on I1. As soon as the pulse is given, the controller starts the forward motion.

Upon completion of the forward move, the controller outputs a pulse for 20 ms and then waits an additional 80 ms before returning to #A for a new cycle.

INSTRUCTION	FUNCTION
#A	Label
AI1	Wait for input 1
PR 6370	Distance
SP 3185	Speed
BGX	Start Motion
AMX	After motion is complete
SB1	Set output bit 1
WT 20	Wait 20 ms
CB1	Clear output bit 1
WT 80	Wait 80 ms
JP #A	Repeat the process

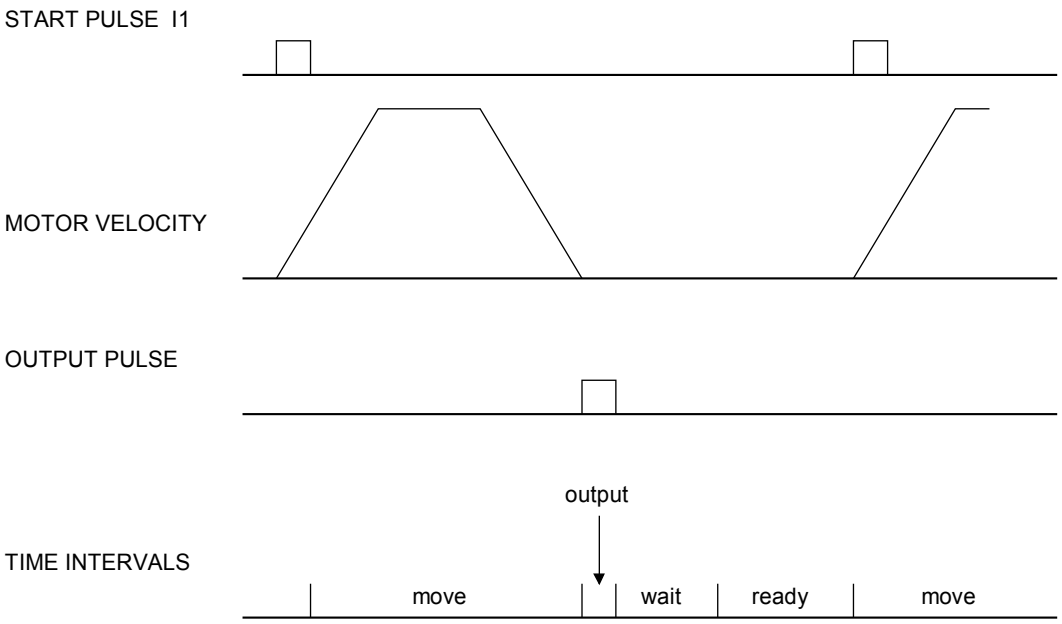


Figure 7.1: Motor Velocity and the Associated Input/Output signals

Speed Control by Joystick

The speed of a motor is controlled by a joystick. The joystick produces a signal in the range between -10V and +10V. The objective is to drive the motor at a speed proportional to the input voltage.

Assume that a full voltage of 10 Volts must produce a motor speed of 3000 rpm with an encoder resolution of 1000 lines or 4000 count/rev. This speed equals:

3000 rpm = 50 rev/sec = 200000 count/sec

The program reads the input voltage periodically and assigns its value to the variable VIN. To get a speed of 200,000 ct/sec for 10 volts, we select the speed as:

Speed = 20000 x VIN

The corresponding velocity for the motor is assigned to the VEL variable.

Instruction

```
#A
JG0
BGX
#B
VIN=@AN[1]
VEL=VIN*20000
JG VEL
JP #B
EN
```

Position Control by Joystick

This system requires the position of the motor to be proportional to the joystick angle. Furthermore, the ratio between the two positions must be programmable. For example, if the control ratio is 5:1, it implies that when the

joystick voltage is 5 Volts, corresponding to 1028 counts, the required motor position must be 5120 counts. The variable V3 changes the position ratio.

```
INSTRUCTION         FUNCTION
      #A                  Label
      V3=5                Initial position ratio
      DP0                 Define the starting position
      JG0                 Set motor in jog mode as zero
      BGX                 Start
      #B
      VIN=@AN[1]          Read analog input
      V2=V1*V3            Compute the desired position
      V4=V2-_TPX-         Find the following error
_TEX
      V5=V4*20            Compute a proportional speed
      JG V5               Change the speed
      JP #B               Repeat the process
      EN                  End
```

Backlash Compensation by Sampled Dual-Loop

The continuous dual loop, enabled by the DV1 function is an effective way to compensate for backlash. In some cases, however, when the backlash magnitude is large, it may be difficult to stabilize the system. In those cases, it may be easier to use the sampled dual loop method described below.

This design example addresses the basic problems of backlash in motion control systems. The objective is to control the position of a linear slide precisely. The slide is to be controlled by a rotary motor, which is coupled to the slide by a lead screw. Such a lead screw has a backlash of 4 micron, and the required position accuracy is for 0.5 micron.

The basic dilemma is where to mount the sensor. If you use a rotary sensor, you get a 4 micron backlash error. On the other hand, if you use a linear encoder, the backlash in the feedback loop will cause oscillations due to instability.

An alternative approach is the dual-loop, where we use two sensors, rotary and linear. The rotary sensor assures stability (because the position loop is closed before the backlash) whereas the linear sensor provides accurate load position information. The operation principle is to drive the motor to a given rotary position near the final point. Once there, the load position is read to find the position error and the controller commands the motor to move to a new rotary position which eliminates the position error.

Since the required accuracy is 0.5 micron, the resolution of the linear sensor should preferably be twice finer. A linear sensor with a resolution of 0.25 micron allows a position error of +/-2 counts.

The dual-loop approach requires the resolution of the rotary sensor to be equal or better than that of the linear system. Assuming that the pitch of the lead screw is 2.5mm (approximately 10 turns per inch), a rotary encoder of 2500 lines per turn or 10,000 count per revolution results in a rotary resolution of 0.25 micron. This results in equal resolution on both linear and rotary sensors.

To illustrate the control method, assume that the rotary encoder is used as a feedback for the X-axis, and that the linear sensor is read and stored in the variable LINPOS. Further assume that at the start, both the position of X and the value of LINPOS are equal to zero. Now assume that the objective is to move the linear load to the position of 1000.

The first step is to command the X motor to move to the rotary position of 1000. Once it arrives we check the position of the load. If, for example, the load position is 980 counts, it implies that a correction of 20 counts must be made. However, when the X-axis is commanded to be at the position of 1000, suppose that the actual position is

only 995, implying that X has a position error of 5 counts, which will be eliminated once the motor settles. This implies that the correction needs to be only 15 counts, since 5 counts out of the 20 would be corrected by the X-axis. Accordingly, the motion correction should be:

Correction = Load Position Error - Rotary Position Error

The correction can be performed a few times until the error drops below +/-2 counts. Often, this is performed in one correction cycle.

Example:

INSTRUCTION	FUNCTION
#A	Label
DP0	Define starting positions as zero
LINPOS=0	
PR 1000	Required distance
BGX	Start motion
#B	
AMX	Wait for completion
WT 50	Wait 50 msec
LINPOS = _DEX	Read linear position
ERR=1000-LINPOS-_TEX	Find the correction
JP #C,@ABS[ERR]<2	Exit if error is small
PR ERR	Command correction
BGX	
JP #B	Repeat the process
#C	
EN	

Chapter 8 Hardware & Software Protection

Introduction

The DMC-30000 provides several hardware and software features to check for error conditions and to inhibit the motor on error. These features help protect the various system components from damage.

WARNING: Machinery in motion can be dangerous! It is the responsibility of the user to design effective error handling and safety protection as part of the machine. Since the DMC-30000 is an integral part of the machine, the engineer should design his overall system with protection against a possible component failure on the DMC-30000. Galil shall not be liable or responsible for any incidental or consequential damages.

Hardware Protection

The DMC-30000 includes hardware input and output protection lines for various error and mechanical limit conditions. These include:

Output Protection Lines

Amp Enable

This signal goes low when the motor off command is given, when the position error exceeds the value specified by the Error Limit (ER) command, or when off-on-error condition is enabled (OE1) and the abort command is given. Each axis amplifier has separate amplifier enable lines. This signal also goes low when the watch-dog timer is activated, or upon reset.

Error Output

The error output is a TTL signal which indicates an error condition in the controller. This signal is available on the interconnect module as ERR. When the error signal is low, this indicates an error condition and the Error Light on the controller will be illuminated. For details on the reasons why the error output would be active see Error Light (Red LED) in Chapter 9.

Input Protection Lines

General Abort

A low input stops commanded motion instantly without a controlled deceleration. For any axis in which the Off-On-Error function is enabled, the amplifiers will be disabled. This could cause the motor to 'coast' to a stop. If the Off-On-Error function is not enabled, the motor will instantaneously stop and servo at the current position. The Off-On-Error function is further discussed in this chapter.

The Abort input by default will also halt program execution; this can be changed by changing the 5th field of the CN command. See the CN command in the command reference for more information.

ELO (Electronic Lock Out)

Used in conjunction with Galil amplifiers, this input allows the user the shutdown the amplifier at a hardware level. For more detailed information on how specific Galil amplifiers behave when the ELO is triggered, see Error: Reference source not found in the Appendices.

Forward Limit Switch

Low input inhibits motion in forward direction. If the motor is moving in the forward direction when the limit switch is activated, the motion will decelerate and stop. In addition, if the motor is moving in the forward direction, the controller will automatically jump to the limit switch subroutine, #LIMSWI (if such a routine has been written by the user). The CN command can be used to change the polarity of the limit switches. The OE command can also be configured so that the axis will be disabled upon the activation of a limit switch.

Reverse Limit Switch

Low input inhibits motion in reverse direction. If the motor is moving in the reverse direction when the limit switch is activated, the motion will decelerate and stop. In addition, if the motor is moving in the reverse direction, the controller will automatically jump to the limit switch subroutine, #LIMSWI (if such a routine has been written by the user). The CN command can be used to change the polarity of the limit switches. The OE command can also be configured so that the axis will be disabled upon the activation of a limit switch.

Software Protection

The DMC-30000 provides a programmable error limit as well as encoder failure detection. It is recommended that both the position error and encoder failure detection be used when running servo motors with the DMC-30000. Along with position error and encoder failure detection, then DMC-30000 has the ability to have programmable software limit.

Position Error

The error limit can be set for any number between 0 and 2147483647 using the ER n command. The default value for ER is 16384.

Example:

```
ER 200          Set X-axis error limit for 200
```

The units of the error limit are quadrature counts. The error is the difference between the command position and actual encoder position. If the absolute value of the error exceeds the value specified by ER, the controller will generate several signals to warn the host system of the error condition. These signals include:

Signal or Function	State if Error Occurs
# POSERR	Jumps to automatic excess position error subroutine
Error Light	Turns on
OE Function	Shuts motor off if OE1 or OE3
AEN Output Line	Switches to Motor Off state

The Jump on Condition statement is useful for branching on a given error within a program. The position error of X,Y,Z and W can be monitored during execution using the TE command.

Encoder Failure detection

The encoder failure detection on the controller operates based upon two factors that are user settable, a threshold of motor command output (OV), a time above that threshold (OT) in which there is no more than 4 counts of change on the encoder input for that axis. The encoder failure detection is activated with the OA command. When an encoder failure is detected and OA is set to 1 for that axis, the same conditions will occur as a position error.

Conditions for proper operation of Encoder Failure detection

- The axis must have a non-zero KI setting order to detect an encoder failure when the axis is not profiling.
- The IL command must be set to a value greater than the OV setting
- The TL command must be set to a value greater than the OV setting

Example:

The A axis is setup with the following settings for encoder failure detection:

```
OA 1
OT 500
OV 3
OE 1
ER 1000
```

The A axis is commanded to move 300 counts, but the B channel on the encoder has failed and no longer operates. Because the ER setting is greater than the commanded move, the error will not be detected by using the OE and ER commands, but this condition will be detected as a encoder failure. When the axis is commanded to move a 300 counts, the position error will cause the motor command voltage to be increased to a value that will be greater than the OV value, 3 volts in this case. Once the motor command output is greater than the OV threshold for more than than the 500ms defined by the OT command AND there has been less than 4 counts of change on the encoder, then the controller will turn off that axis due to an encoder failure. The motor will have moved some distance during this operation, but it will be shut down before a full runaway condition occurs.

Using Encoder Failure to detect a hard stop or stalled motor

The encoder failure detection can also be used to detect when an axis is up against a hard stop. In this scenario the motor command will be commanded above the OV threshold, but because the motor is not moving the controller will detect this scenario as an encoder failure.

Programmable Position Limits

The DMC-30000 provides programmable forward and reverse position limits. These are set by the BL and FL software commands. Once a position limit is specified, the DMC-30000 will not accept position commands beyond the limit. Motion beyond the limit is also prevented.

Example:

```
DP 0              Define Position
BL -2000          Set Reverse position limit
FL 2000           Set Forward position limit
JG 2000           Jog
BG X              Begin
```

(motion stops at forward limits)

Off-On-Error

The DMC-30000 controller has a built in function which can turn off the motors under certain error conditions. This function is known as 'Off-On-Error". To activate the OE function for each axis, specify 1, 2 or 3 for that axis. To disable this function, specify 0 for the axes. When this function is enabled, the specified motor will be disabled under the following 3 conditions:

1. The position error for the specified axis exceeds the limit set with the command, ER
2. A hardware limit is reached
3. The abort command is given
4. The abort input is activated with a low signal.

NOTE: If the motors are disabled while they are moving, they may 'coast' to a stop because they are no longer under servo control.

To re-enable the system, use the Reset (RS) or Servo Here (SH) command.

Examples:

```
OE 1          Enable off-on-error
```

Automatic Error Routine

The #POSERR label causes the statements following to be automatically executed if error on any axis exceeds the error limit specified by ER, a encoder failure is detected, or the abort input is triggered. The error routine must be closed with the RE command. The RE command returns from the error subroutine to the main program.

NOTE: The Error Subroutine will be entered again unless the error condition is cleared.

Example:

```
#A;JP #A;EN        "Dummy" program
#POSERR            Start error routine on error
MG "error"         Send message
SB 1               Fire relay
STX                Stop motor
AMX                After motor stops
SHX                Servo motor here to clear error
RE                 Return to main program
```

Limit Switch Routine

The DMC-30000 provides forward and reverse limit switches which inhibit motion in the respective direction. There is also a special label for automatic execution of a limit switch subroutine. The #LIMSWI label specifies the start of the limit switch subroutine. This label causes the statements following to be automatically executed if any limit switch is activated and that axis motor is moving in that direction. The RE command ends the subroutine.

The state of the forward and reverse limit switches may also be tested during the jump-on-condition statement. The _LR condition specifies the reverse limit and _LF specifies the forward limit. X,Y,Z, or W following LR or LF specifies the axis. The CN command can be used to configure the polarity of the limit switches.

Limit Switch Example:

```
#A;JP #A;EN        Dummy Program
#LIMSWI            Limit Switch Utility
V1=_LFX            Check if forward limit
V2=_LRX            Check if reverse limit
```

```
      JP#LF,V1=0          Jump to #LF if forward
      JP#LR,V2=0          Jump to #LR if reverse
      JP#END              Jump to end
      #LF                 #LF
      MG “FORWARD         Send message
LIMIT”
      STX;AMX             Stop motion
      PR-                 Move in reverse
1000;BGX;AMX
      JP#END              End
      #LR                 #LR
      MG “REVERSE         Send message
LIMIT”
      STX;AMX             Stop motion
      PR1000;BGX;         Move forward
AMX
      #END                End
      RE                  Return to main program
```

Chapter 9 Troubleshooting

Overview

The following discussion may help you get your system to work.

Potential problems have been divided into groups as follows:

1. Installation
2. Stability and Compensation
3. Operation
4. Error Light (Red LED)

The various symptoms along with the cause and the remedy are described in the following tables.

Installation

SYMPTOM	DIAGNOSIS	CAUSE	REMEDY
Motor runs away with no connections from controller to amplifier input.	Adjusting offset causes the motor to change speed.	1. Amplifier has an internal offset. 2. Damaged amplifier.	Adjust amplifier offset. Amplifier offset may also be compensated by use of the offset configuration on the controller (see the OF command). Replace amplifier.
Motor is enabled even when MO command is given	The SH command disables the motor	1. The amplifier requires the a different Amplifier Enable setting on the Interconnect Module	Refer to Chapter 3 or contact Galil.
Unable to read main or auxiliary encoder input.	The encoder does not work when swapped with another encoder input.	1. Wrong encoder connections. 2. Encoder is damaged 3. Encoder configuration incorrect.	Check encoder wiring. For single ended encoders (CHA and CHB only) do not make any connections to the CHA- and CHB- inputs. Replace encoder Check CE command

Unable to read main or auxiliary encoder input.	The encoder works correctly when swapped with another encoder input.	1. Wrong encoder connections. 2. Encoder configuration incorrect. 3. Encoder input or controller is damaged	Check encoder wiring. For single ended encoders (MA+ and MB+ only) do not make any connections to the MA- and MB- inputs. Check CE command Contact Galil
Encoder Position Drifts	Swapping cables fixes the problem	1. Poor Connections / intermittent cable	Review all connections and connector contacts.
Encoder Position Drifts	Significant noise can be seen on MA+ and / or MB+ encoder signals	1. Noise	Shield encoder cables Avoid placing power cables near encoder cables Avoid Ground Loops Use differential encoders Use +/-12V encoders

Stability

SYMPTOM	DIAGNOSIS	CAUSE	REMEDY
Servo motor runs away when the loop is closed.	Reversed Motor Type corrects situation (MT -1)	1. Wrong feedback polarity.	Reverse Motor or Encoder Wiring (remember to set Motor Type back to default value: MT 1)
Motor oscillates.		2. Too high gain or too little damping.	Decrease KI and KP. Increase KD.

Operation

SYMPTOM	DIAGNOSIS	CAUSE	REMEDY
Controller rejects commands.	Response of controller from TC1 diagnoses error.	1. Anything	Correct problem reported by TC1
Motor Doesn't Move	Response of controller from TC1 diagnoses error.	2. Anything	Correct problem reported by SC

Error Light (Red LED)

The red error LED has multiple meanings for Galil controllers. Here is a list of reasons the error light will come on and possible solutions:

Under Voltage

If the controller is not receiving enough voltage to power up.

Under Current

If the power supply does not have enough current, the red LED will cycle on and off along with the green power LED.

Position Error

If any axis that is set up as a servo (MT command) has a position error value (TE) that exceeds the error limit (ER) - the error light will come on to signify there is an axis that has exceeded the position error limit. Use a DP*=0 to set all encoder positions to zero or a SH (Servo Here) command to eliminate position error.

Invalid Firmware

If the controller is interrupted during a firmware update or an incorrect version of firmware is installed - the error light will come on. The prompt will show up as a greater than sign ">" instead of the standard colon ":" prompt. Use GalilTools software to install the correct version of firmware to fix this problem.

Self Test

During the first few seconds of power up, it is normal for the red LED to turn on while it is performing a self test. If the self test detects a problem such as corrupted memory or damaged hardware - the error light will stay on to signal a problem with the board. To fix this problem, a Master Reset may be required. The Master Reset will set the controller back to factory default conditions so it is recommended that all motor and I/O cables be removed for safety while performing the Master Reset. Cables can be plugged back in after the correct settings have been loaded back to the controller (when necessary). To perform a Master Reset - find the jumper location labeled MR or MR on the controller and put a jumper across the two pins. Power up with the jumper installed. The Self-Test will take slightly longer - up to 5seconds. After the error light shuts off, it is safe to power down and remove the Master Reset jumper. If performing a Master Reset does not get rid of the error light, the controller may need to be sent back to the factory to be repaired. Contact Galil for more information.

Chapter 10 Theory of Operation

Overview

The following discussion covers the operation of motion control systems. A typical motion control system consists of the elements shown in Figure 10.1.

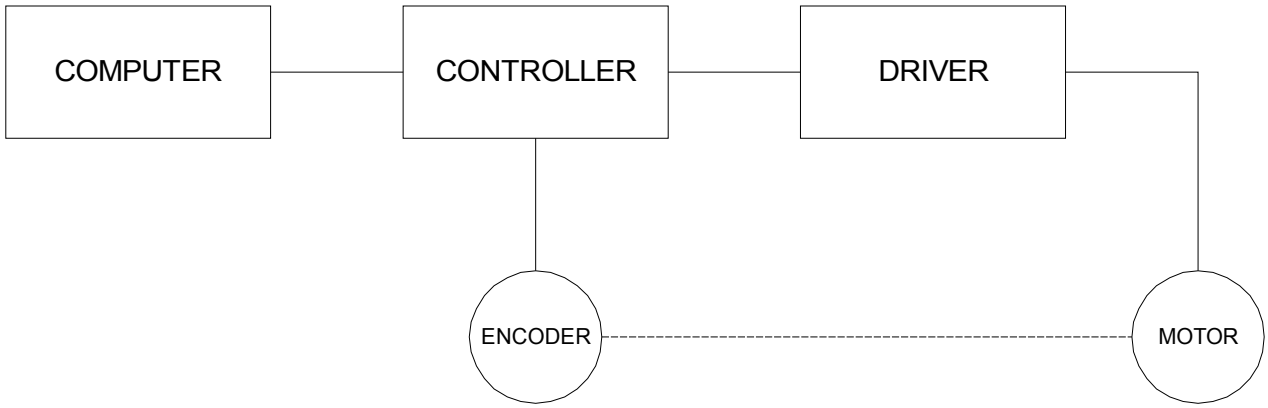


Figure 10.1: Elements of Servo Systems

The operation of such a system can be divided into three levels, as illustrated in Figure 10.2. The levels are:

1. Closing the Loop
2. Motion Profiling
3. Motion Programming

The first level, the closing of the loop, assures that the motor follows the commanded position. This is done by closing the position loop using a sensor. The operation at the basic level of closing the loop involves the subjects of modeling, analysis, and design. These subjects will be covered in the following discussions.

The motion profiling is the generation of the desired position function. This function, R(t), describes where the motor should be at every sampling period. Note that the profiling and the closing of the loop are independent functions. The profiling function determines where the motor should be and the closing of the loop forces the motor to follow the commanded position

The highest level of control is the motion program. This can be stored in the host computer or in the controller. This program describes the tasks in terms of the motors that need to be controlled, the distances and the speed.

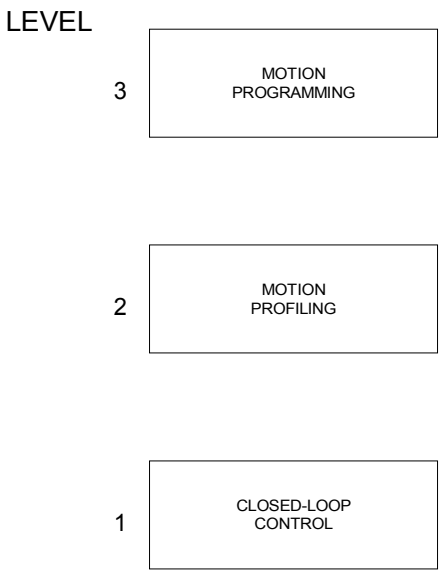


Figure 10.2: Levels of Control Functions

The three levels of control may be viewed as different levels of management. The top manager, the motion program, may specify the following instruction, for example.

```
PR 6000
SP 20000
AC 200000
BG X
EN
```

This program corresponds to the velocity profiles shown in Figure 10.3. Note that the profiled positions show where the motors must be at any instant of time.

Finally, it remains up to the servo system to verify that the motor follows the profiled position by closing the servo loop.

The following section explains the operation of the servo system. First, it is explained qualitatively, and then the explanation is repeated using analytical tools for those who are more theoretically inclined.

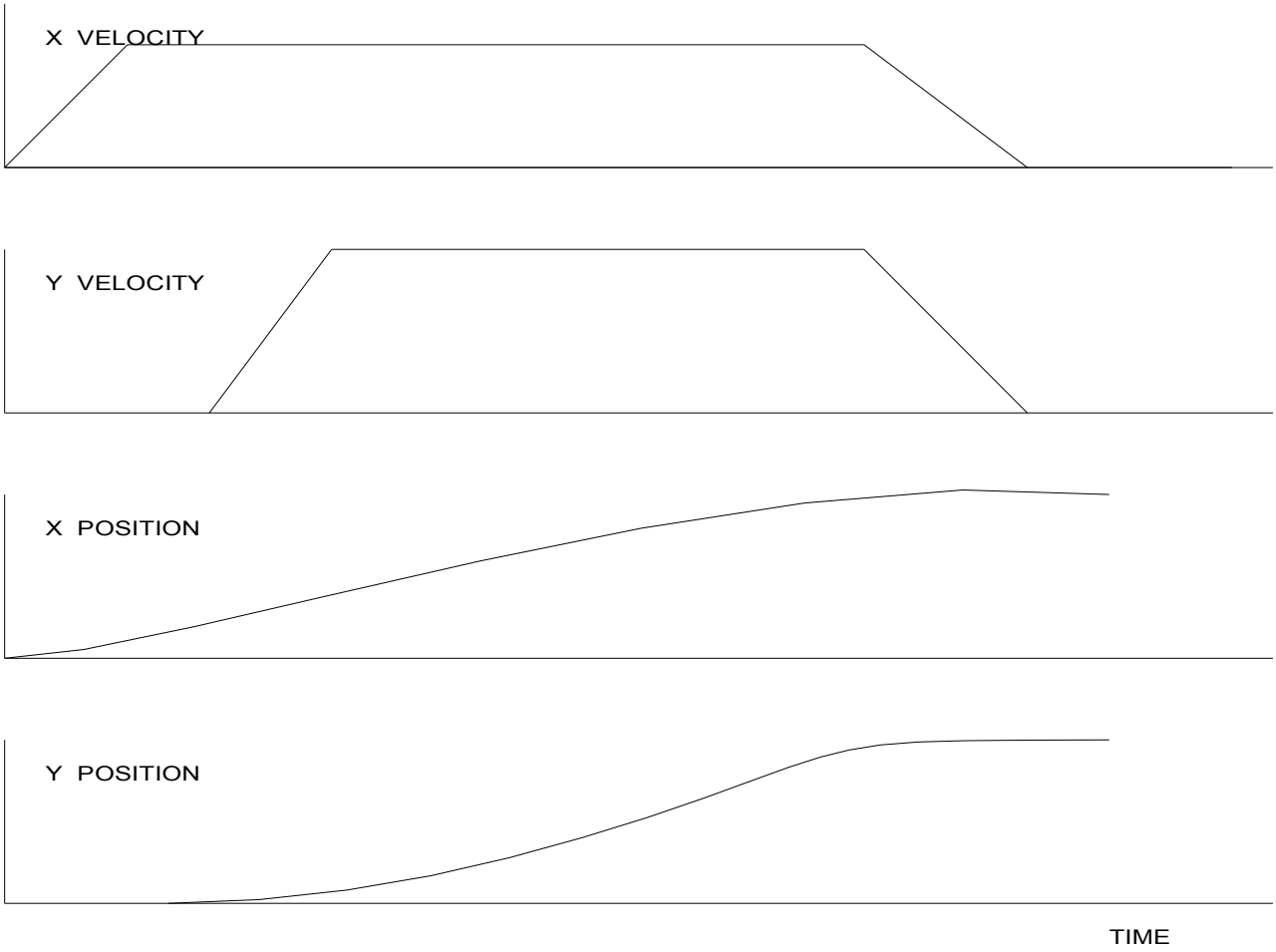


Figure 10.3: Velocity and Position Profiles

Operation of Closed-Loop Systems

To understand the operation of a servo system, we may compare it to a familiar closed-loop operation, adjusting the water temperature in the shower. One control objective is to keep the temperature at a comfortable level, say 90 degrees F. To achieve that, our skin serves as a temperature sensor and reports to the brain (controller). The brain compares the actual temperature, which is called the feedback signal, with the desired level of 90 degrees F. The difference between the two levels is called the error signal. If the feedback temperature is too low, the error is positive, and it triggers an action which raises the water temperature until the temperature error is reduced sufficiently.

The closing of the servo loop is very similar. Suppose that we want the motor position to be at 90 degrees. The motor position is measured by a position sensor, often an encoder, and the position feedback is sent to the controller. Like the brain, the controller determines the position error, which is the difference between the commanded position of 90 degrees and the position feedback. The controller then outputs a signal that is proportional to the position error. This signal produces a proportional current in the motor, which causes a motion until the error is reduced. Once the error becomes small, the resulting current will be too small to overcome the friction, causing the motor to stop.

The analogy between adjusting the water temperature and closing the position loop carries further. We have all learned the hard way, that the hot water faucet should be turned at the "right" rate. If you turn it too slowly, the temperature response will be slow, causing discomfort. Such a slow reaction is called over-damped response.

The results may be worse if we turn the faucet too fast. The overreaction results in temperature oscillations. When the response of the system oscillates, we say that the system is unstable. Clearly, unstable responses are bad when we want a constant level.

What causes the oscillations? The basic cause for the instability is a combination of delayed reaction and high gain. In the case of the temperature control, the delay is due to the water flowing in the pipes. When the human reaction is too strong, the response becomes unstable.

Servo systems also become unstable if their gain is too high. The delay in servo systems is between the application of the current and its effect on the position. Note that the current must be applied long enough to cause a significant effect on the velocity, and the velocity change must last long enough to cause a position change. This delay, when coupled with high gain, causes instability.

This motion controller includes a special filter which is designed to help the stability and accuracy. Typically, such a filter produces, in addition to the proportional gain, damping and integrator. The combination of the three functions is referred to as a PID filter.

The filter parameters are represented by the three constants KP, KI and KD, which correspond to the proportional, integral and derivative term respectively.

The damping element of the filter acts as a predictor, thereby reducing the delay associated with the motor response.

The integrator function, represented by the parameter KI, improves the system accuracy. With the KI parameter, the motor does not stop until it reaches the desired position exactly, regardless of the level of friction or opposing torque.

The integrator also reduces the system stability. Therefore, it can be used only when the loop is stable and has a high gain.

The output of the filter is applied to a digital-to-analog converter (DAC). The resulting output signal in the range between +10 and -10 Volts is then applied to the amplifier and the motor.

The motor position, whether rotary or linear is measured by a sensor. The resulting signal, called position feedback, is returned to the controller for closing the loop.

The following section describes the operation in a detailed mathematical form, including modeling, analysis and design.

System Modeling

The elements of a servo system include the motor, driver, encoder and the controller. These elements are shown in Figure 10.4. The mathematical model of the various components is given below.

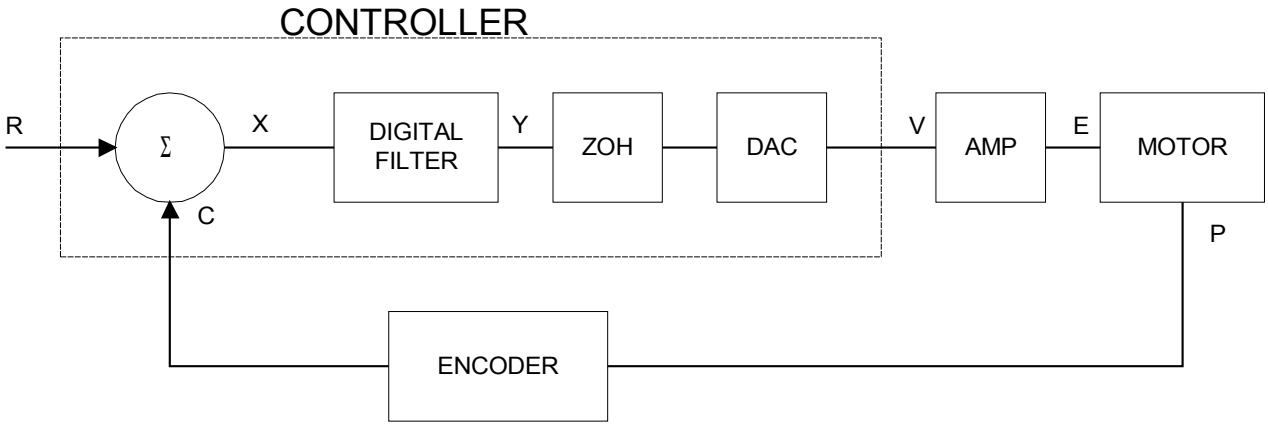


Figure 10.4: Functional Elements of a Motion Control System

Motor-Amplifier

The motor amplifier may be configured in three modes:

1. Voltage Drive
2. Current Drive
3. Velocity Loop

The operation and modeling in the three modes is as follows:

Voltage Drive

The amplifier is a voltage source with a gain of K_V [V/V]. The transfer function relating the input voltage, V, to the motor position, P, is

$$P/V = K_V / \left[K_t S (ST_m + 1)(ST_e + 1) \right]$$

where

$$T_m = RJ/K_t^2 \quad [s]$$

and

$$T_e = L/R \quad [s]$$

and the motor parameters and units are

K_t	Torque constant [Nm/A]
R	Armature Resistance Ω
J	Combined inertia of motor and load [kg.m^2]
L	Armature Inductance [H]

When the motor parameters are given in English units, it is necessary to convert the quantities to MKS units. For example, consider a motor with the parameters:

K_t = 14.16 oz - in/A = 0.1 Nm/A

$R = 2\ \Omega$

$J = 0.0283$ oz-in-$s^2 = 2 * 10^{-4}$ kg . m^2

$L = 0.004H$

Then the corresponding time constants are

$T_m = 0.04$ sec

and

$T_e = 0.002$ sec

Assuming that the amplifier gain is $K_v = 4$, the resulting transfer function is

$P/V = 40/[s(0.04s+1)(0.002s+1)]$

Current Drive

The current drive generates a current I, which is proportional to the input voltage, V, with a gain of K_a. The resulting transfer function in this case is

$P/V = K_a K_t / Js^2$

where Kt and J are as defined previously. For example, a current amplifier with $K_a = 2$ A/V with the motor described by the previous example will have the transfer function:

$P/V = 1000/s^2$ [rad/V]

If the motor is a DC brushless motor, it is driven by an amplifier that performs the commutation. The combined transfer function of motor amplifier combination is the same as that of a similar brush motor, as described by the previous equations.

Velocity Loop

The motor driver system may include a velocity loop where the motor velocity is sensed by a tachometer and is fed back to the amplifier. Such a system is illustrated in Figure 10.5. Note that the transfer function between the input voltage V and the velocity ω is:

$\omega /V = [K_a K_t/Js]/[1+K_a K_t K_g/Js] = 1/[K_g(sT_1+1)]$

where the velocity time constant, T_1, equals

$T_1 = J/K_a K_t K_g$

This leads to the transfer function

$P/V = 1/[K_g s(sT_1+1)]$

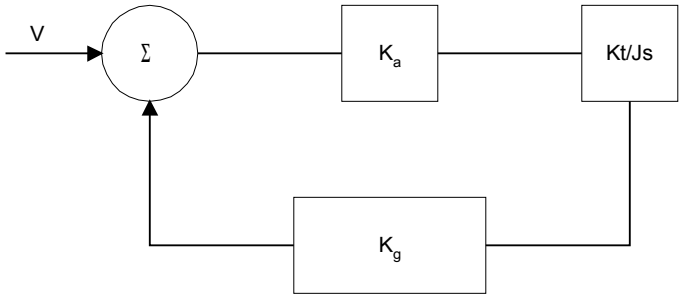


Figure 10.5: Elements of velocity loops

The resulting functions derived above are illustrated by the block diagram of Figure 10.6.

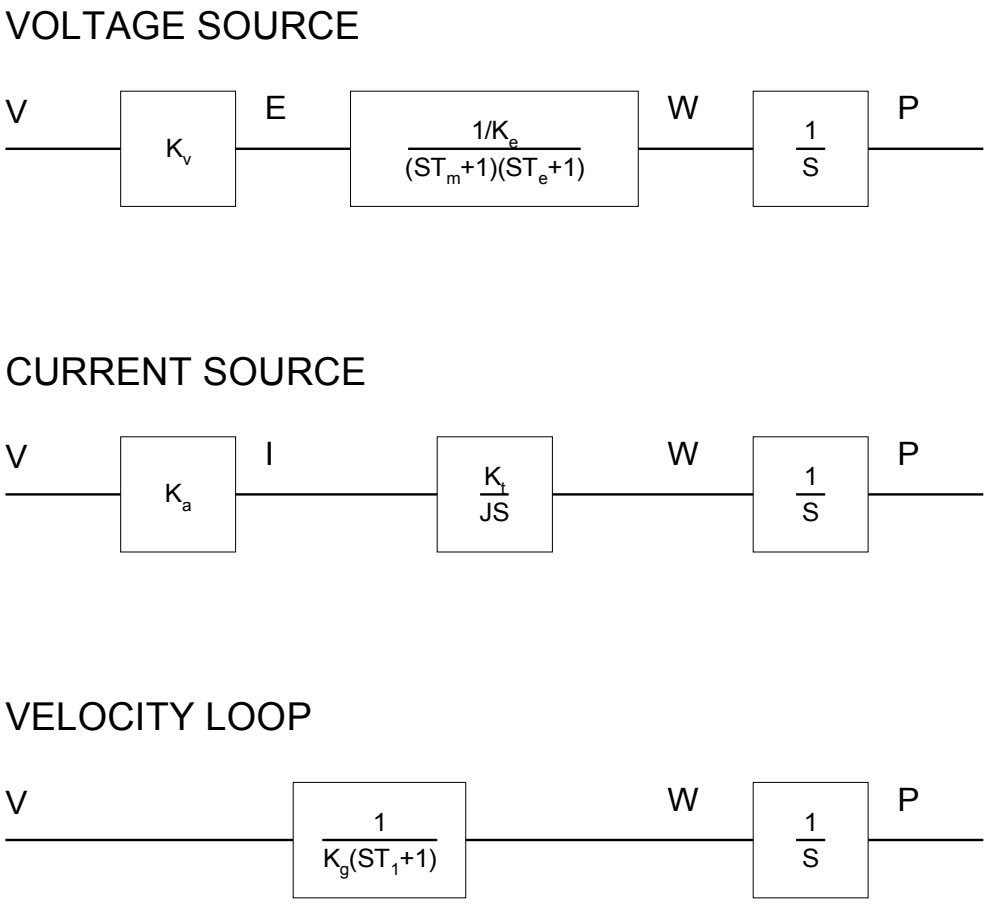


Figure 10.6: Mathematical model of the motor and amplifier in three operational modes

Encoder

The encoder generates N pulses per revolution. It outputs two signals, Channel A and B, which are in quadrature. Due to the quadrature relationship between the encoder channels, the position resolution is increased to 4N quadrature counts/rev.

The model of the encoder can be represented by a gain of

$K_f = 4N/2\pi$ [count/rad]

For example, a 1000 lines/rev encoder is modeled as

$K_f = 638$

DAC

The DAC or D-to-A converter converts a 16-bit number to an analog voltage. The input range of the numbers is 65536 and the output voltage range is +/-10V or 20V. Therefore, the effective gain of the DAC is

K= 20/65536 = 0.0003 [V/count]

Digital Filter

The digital filter has three element in series: PID, low-pass and a notch filter. The transfer function of the filter. The transfer function of the filter elements are:

PID $D(z) = \frac{K(Z - A)}{Z} + \frac{CZ}{Z - 1}$

Low-pass $L(z) = \frac{1 - B}{Z - B}$

Notch $N(z) = \frac{(Z - z)(Z - \bar{z})}{(Z - p)(Z - \bar{p})}$

The filter parameters, K, A, C and B are selected by the instructions KP, KD, KI and PL, respectively. The relationship between the filter coefficients and the instructions are:

K = (KP + KD)
A = KD/(KP + KD)
C = KI
B = PL

The PID and low-pass elements are equivalent to the continuous transfer function G(s).

G(s) = (P + sD + I/s) ∗ a / (s + a)

where,

P = KP
D = T · KD
I = KI/T

$a = \frac{1}{T} ln\left(\frac{1}{B}\right)$

where T is the sampling period, and B is the pole setting

For example, if the filter parameters of the DMC-30000 are

KP = 16
KD = 144
KI = 2
PL = 0.75
T = 0.001 s

the digital filter coefficients are

K = 160
A = 0.9
C = 2
a = 250 rad/s

and the equivalent continuous filter, G(s), is

G(s) = [16 + 0.144s + 2000/s] ∗ 250/ (s+250)

The notch filter has two complex zeros, z and $\bar{z}$, and two complex poles, p and $\bar{p}$.

The effect of the notch filter is to cancel the resonance affect by placing the complex zeros on top of the resonance poles. The notch poles, P and p, are programmable and are selected to have sufficient damping. It is best to select

the notch parameters by the frequency terms. The poles and zeros have a frequency in Hz, selected by the command NF. The real part of the poles is set by NB and the real part of the zeros is set by NZ.

The most simple procedure for setting the notch filter, identify the resonance frequency and set NF to the same value. Set NB to about one half of NF and set NZ to a low value between zero and 5.

ZOH

The ZOH, or zero-order-hold, represents the effect of the sampling process, where the motor command is updated once per sampling period. The effect of the ZOH can be modeled by the transfer function

$H(s) = 1/(1+sT/2)$

If the sampling period is T = 0.001, for example, H(s) becomes:

$H(s) = 2000/(s+2000)$

However, in most applications, H(s) may be approximated as one.

This completes the modeling of the system elements. Next, we discuss the system analysis.

System Analysis

To analyze the system, we start with a block diagram model of the system elements. The analysis procedure is illustrated in terms of the following example.

Consider a position control system with the DMC-30000 controller and the following parameters:

$K_t = 0.1$	Nm/A	Torque constant
$J = 2 * 10^{-4}$	$kg.m^2$	System moment of inertia
$R = 2$	Ω	Motor resistance
$K_a = 4$	Amp/Volt	Current amplifier gain
KP = 12.5		Digital filter gain
KD = 245		Digital filter zero
KI = 0		No integrator
N = 500	Counts/rev	Encoder line density
T = 1	ms	Sample period

The transfer function of the system elements are:

Motor

$M(s) = P/I = K_t/Js^2 = 500/s^2$ [rad/A]

Amp

$K_a = 4$ [Amp/V]

DAC

$K_d = 0.0003$ [V/count]

Encoder

$K_f = 4N/2\pi = 318$ [count/rad]

ZOH

2000/(s+2000)

Digital Filter

KP = 12.5, KD = 245, T = 0.001

Therefore,

$$D(z) = 1030\ (z-0.95)/Z$$

Accordingly, the coefficients of the continuous filter are:

$$P = 50$$

$$D = 0.98$$

The filter equation may be written in the continuous equivalent form:

$$G(s) = 50 + 0.98s = .098\ (s+51)$$

The system elements are shown in Figure 10.7.

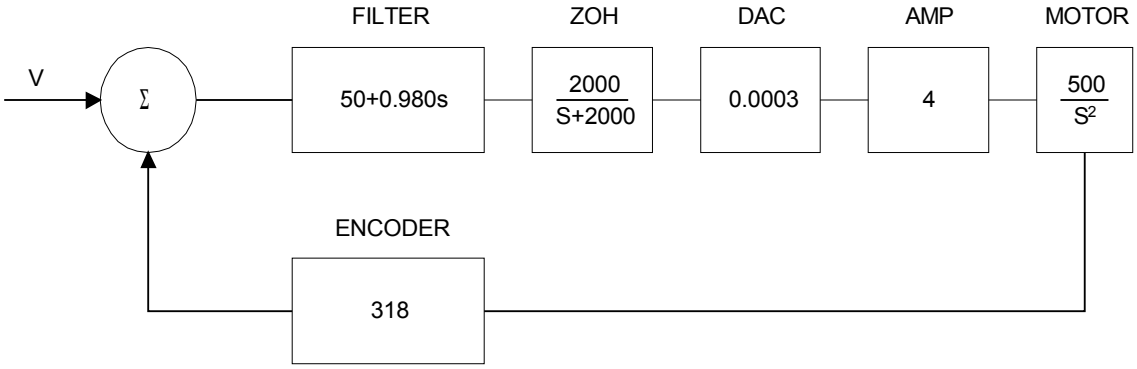


Figure 10.7: Mathematical model of the control system

The open loop transfer function, A(s), is the product of all the elements in the loop.

$$A(s) = 390{,}000\ (s+51)/[s^2(s+2000)]$$

To analyze the system stability, determine the crossover frequency, ω_c at which A(j ω_c) equals one. This can be done by the Bode plot of A(j ω_c), as shown in Figure 10.8.

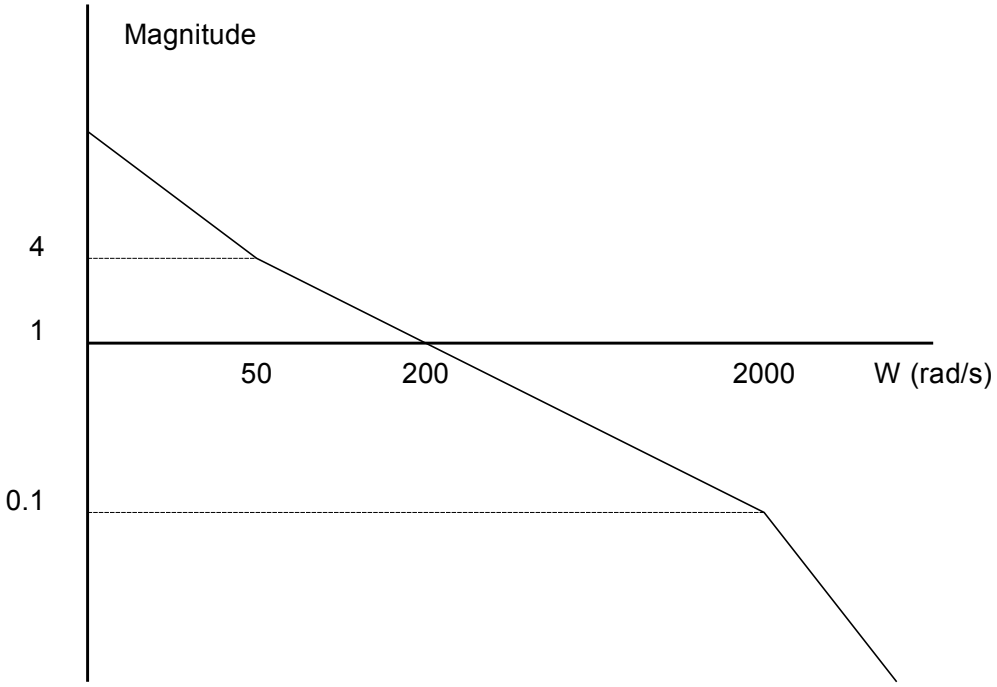


Figure 10.8: Bode plot of the open loop transfer function

For the given example, the crossover frequency was computed numerically resulting in 200 rad/s.

Next, we determine the phase of A(s) at the crossover frequency.

$$A(j200) = 390{,}000\ (j200+51)/[(j200)^2\ .\ (j200 + 2000)]$$

$$\alpha = \text{Arg}[A(j200)] = \tan^{-1}(200/51) - 180° - \tan^{-1}(200/2000)$$

$$\alpha = 76° - 180° - 6° = -110°$$

Finally, the phase margin, PM, equals

$$PM = 180° + \alpha = 70°$$

As long as PM is positive, the system is stable. However, for a well damped system, PM should be between 30° and 45°. The phase margin of 70° given above indicated over-damped response.

Next, we discuss the design of control systems.

System Design and Compensation

The closed-loop control system can be stabilized by a digital filter, which is preprogrammed in the DMC-30000 controller. The filter parameters can be selected by the user for the best compensation. The following discussion presents an analytical design method.

The Analytical Method

The analytical design method is aimed at closing the loop at a crossover frequency, ω_c, with a phase margin PM. The system parameters are assumed known. The design procedure is best illustrated by a design example.

Consider a system with the following parameters:

$K_t = 0.2$	Nm/A	Torque constant
$J = 2 * 10^{-4}$	$kg.m^2$	System moment of inertia
$R = 2$	Ω	Motor resistance
$K_a = 2$	Amp/Volt	Current amplifier gain
$N = 1000$	Counts/rev	Encoder line density

The DAC of theDMC-30000 outputs +/-10V for a 16-bit command of +/-32768 counts.

The design objective is to select the filter parameters in order to close a position loop with a crossover frequency of $\omega_c = 500$ rad/s and a phase margin of 45 degrees.

The first step is to develop a mathematical model of the system, as discussed in the previous system.

Motor

$$M(s) = P/I = Kt/Js^2 = 1000/s^2$$

Amp

$$K_a = 2 \qquad [Amp/V]$$

DAC

$$K_d = 10/32768 = .0003$$

Encoder

$$K_f = 4N/2\pi = 636$$

ZOH

$$H(s) = 2000/(s+2000)$$

Compensation Filter

$$G(s) = P + sD$$

The next step is to combine all the system elements, with the exception of G(s), into one function, L(s).

$$L(s) = M(s)\, K_a\, K_d\, K_f\, H(s) = 3.17 * 10^6/[s^2(s+2000)]$$

Then the open loop transfer function, A(s), is

$$A(s) = L(s)\, G(s)$$

Now, determine the magnitude and phase of L(s) at the frequency $\omega_c = 500$.

$$L(j500) = 3.17 * 10^6/[(j500)^2\ (j500+2000)]$$

This function has a magnitude of

$$|L(j500)| = 0.00625$$

and a phase

$$\text{Arg}[L(j500))] = -180° - \tan^{-1}[500/2000] = -194°$$

G(s) is selected so that A(s) has a crossover frequency of 500 rad/s and a phase margin of 45 degrees. This requires that

$$|A(j500)| = 1$$
$$\text{Arg}\ [A(j500)] = -135°$$

However, since

$$A(s) = L(s)\, G(s)$$

then it follows that G(s) must have magnitude of

$$|G(j500)| = |A(j500)/L(j500)| = 160$$

and a phase

$$\arg\ [G(j500)] = \arg\ [A(j500)] - \arg\ [L(j500)] = -135° + 194° = 59°$$

In other words, we need to select a filter function G(s) of the form

$$G(s) = P + sD$$

so that at the frequency $\omega_c = 500$, the function would have a magnitude of 160 and a phase lead of 59 degrees.

These requirements may be expressed as:

$$|G(j500)| = |P + (j500D)| = 160$$

and

$$\arg\ [G(j500)] = \tan^{-1}[500D/P] = 59°$$

The solution of these equations leads to:

$$P = 160\cos 59° = 82.4$$
$$500D = 160\sin 59° = 137$$

Therefore,

$$D = 0.274$$

and

$$G = 82.4 + 0.274s$$

The function G is equivalent to a digital filter of the form:

$$D(z) = KP + KD(1-z^{-1})$$

where

$$P = KP$$

$$D = KD * T$$

and

$$KD = D/T$$

Assuming a sampling period of T=1ms, the parameters of the digital filter are:

$$KP = 82.4$$

$$KD = 274$$

The DMC-30000 can be programmed with the instruction:

KP 82.4

KD 274

In a similar manner, other filters can be programmed. The procedure is simplified by the following table, which summarizes the relationship between the various filters.

Equivalent Filter Form - DMC-30000

Digital	$D(z) = [K(z-A/z) + Cz/(z-1)] * (1-B)/(Z-B)$
KP, KD, KI, PL	$K = (KP + KD)$
	$A = KD/(KP+KD)$
	$C = KI$
	$B = PL$
Digital	$D(z) = [KP + KD(1-z^{-1}) + KI/2(1-z^{-1})] * (1-PL)/(Z-PL)$
Continuous	$G(s) = (P + Ds + I/s) * a/(s+a)$
PID, T	$P = KP$
	$D = T * KD$
	$I = KI / T$
	$a = 1/T \ln(1/PL)$

Appendices

Electrical Specifications

Servo Control

AO1 and AO2 Amplifier Command:	+/-10 volt analog signal. Resolution 16-bit DAC or 0.0003 volts. 3 mA maximum. Output impedance – 500Ω
MA+,MA-,MB+,MB-,MI+,MI- Encoder and Auxiliary	TTL compatible, but can accept up to +/-12 volts. Quadrature phase on CHA, CHB. Can accept single-ended (A+,B+ only) or differential (A+,A-,B+,B-). Maximum A, B edge rate: 15 MHz. Minimum IDX pulse width: 30 nsec.

Stepper Control

MF2+ MF2- (Step)	Differential (0-3.3 Volts) level at 50% duty cycle. 3,000,000 pulses/sec maximum frequency
MF4+ MF4- (Direction)	Differential (0-3.3 Volts)

Input / Output

Limit Switch Inputs, Home Inputs. DI1 thru DI8 Uncommitted Inputs and Abort Input	2.2K Ω in series with optoisolator. Active high or low requires at least 1.2mA to activate. Once activated, the input requires the current to go below 0.5ma. All Limit Switch and Home inputs use one common voltage (LSCOM) which can accept up to 24 volts. Voltages above 24 volts require an additional resistor.
Analog Inputs:	Standard configuration is 0-5 volts. 12-Bit Analog-to-Digital converter.
DO1 thru DO4 Outputs:	Optoisolated – 4mA sinking (25mA and 500mA options)
DI81, DI82	Auxiliary Encoder Inputs for A (X) axis. Line Receiver Inputs - accepts differential or single ended voltages with voltage range of +/- 12 volts.

Input Power Requirements

Controller Model	Input Voltage Requirement	Current/Power Requirement [1]
DMC-30010	+5 VDC (+/-10%) +12VDC (+/-10%) -12VDC (+/-10%)	0.5 Amps 0.05 Amps 0.05 Amps
DMC-30011	9-48 VDC	3 Watts
DMC-30011(P80V)	20-80 VDC	
DMC-30012	20-80 VDC	5 Watts [2]

1 – Power Requirements the required power with no external connections

2 – Does not include power for the motor. The power supply should be sized based upon load and motor specifications.

NOTE: The DMC-30000 power should never be plugged in HOT. Always power down the power supply before installing or removing the power connector on the controller.

+5, +/-12V Power Output Specifications (DMC-30011 and DMC-30012)

Output Voltage	Tolerance	Max Current Output
+5V	+/- 10%	0.5A
+12V	+/- 10%	10mA
-12V	+/- 10%	10mA

Performance Specifications

Minimum Servo Loop Update Time/Memory:

	Normal
Minimum Servo Loop Update Time:	
DMC-30000	125 μsec
Position Accuracy:	+/-1 quadrature count
Velocity Accuracy:	
Long Term	Phase-locked, better than 0.005%
Short Term	System dependent
Position Range:	+/-2147483647 counts per move
Velocity Range:	Up to 15,000,000 counts/sec servo; 3,000,000 pulses/sec-stepper
Velocity Resolution:	2 counts/sec
Motor Command Resolution:	16 bit or 0.0003 V
Variable Range:	+/-2 billion
Variable Resolution:	1 · 10-4
Number of Variables:	510
Array Size:	16000 elements, 30 arrays
Program Size:	2000 lines x 80 characters

Fast Update Rate Mode

The DMC-30000 can operate with much faster servo update rates than the default of every millisecond. This mode is known as 'fast mode' and allows the controller to operate at an update rate of 62.5usec.

In order to run the DMC-30000 motion controller in fast mode, the fast firmware must be uploaded. This can be done through the GalilTools communication software. The fast firmware is included with the original DMC-30000 utilities.

In order to set the desired update rates, use the command TM.

When the controller is operating with the fast firmware, the following functions are disabled:

Gearing mode
Ecam mode
Pole (PL)
Analog Feedback (AF)
Stepper Motor Operation (MT 2,-2,2.5,-2.5)
Trippoints in thread 2 and 3
Tell Velocity Interrogation Command (TV)
Aux Encoders (TD)
Dual Velocity (DV)
Peak Torque Limit (TK)
Notch Filter (NB, NF, NZ)
PVT Mode (PV, BT)

Ordering Options for the DMC-30000

Overview

The DMC-30000 can be ordered in many different configurations and with different options. This section provides information regarding the different options available on the DMC-30000 motion controller, interconnect modules and internal amplifiers. For information on pricing and how to order your controller with these options, see our DMC-30000 part number generator on our website.

http://www.galilmc.com/products/dmc-300xx-part-number.php

I/O Options

4-20mA – 4-20mA analog inputs

The 4-20mA option converts the analog inputs into 4-20mA analog inputs. This is accomplished by installing 237Ω precision resistors between the analog inputs and ground. The equation for calculating the current is:

$$I_{ma} = 2.11\ V$$

Where I_{ma} = current in mA

V = Voltage reading from DMC-30000

Part number ordering example: DMC-30010-CARD(4-20mA)

LSNK – 25mA Sinking Outputs

The LSNK option modifies the digital outputs on the DMC-30000 to be capable of sinking up to 25mA per output. For detailed information see the 25mA Sinking Optoisolated Outputs (LSNK) section in Chapter 3 Connecting Hardware.

Part number ordering example: DMC-30010-CARD-(LSNK)

LSRC – 25mA Sourcing Outputs

The LSRC option modifies the digital outputs on the DMC-30000 to be capable of sourcing up to 25mA per output. For detailed information see the 25mA Sourcing Optoisolated Outputs (LSRC) section in Chapter 3 Connecting Hardware.

Part number ordering example: DMC-30010-CARD(LSRC)

HSRC – 500mA Sourcing Outputs

The HSRC option modifies the digital outputs on the DMC-30000 to be capable of sourcing up to 500mA per output. For detailed information see the 500mA Sourcing Optoisolated Outputs (HSRC) section in Chapter 3 Connecting Hardware.

Part number ordering example: DMC-30010-CARD(HSRC)

HSNK – 500mA Sinking Outputs

The HSNK option modifies the digital outputs on the DMC-30000 to be capable of sinking up to 500mA per output. For detailed information see the 500mA Sinking Optoisolated Outputs (HSNK) section in Chapter 3 Connecting Hardware.

Part number ordering example: DMC-30010-CARD(HSNK)

DMC-31000 – Sin/Cos and 16 bit Analog Inputs

The DMC-31000 provides 16-bit configurable +/-10V analog inputs in place of the standard 12-bit 0-5V analog inputs. See the Analog Inputs section in Chapter 3 Connecting Hardware for more information.

Part number ordering example: DMC-31012-BOX

Feedback Options

TRES – Encoder Termination Resistors

The TRES option provides termination resistors on all of the main and auxiliary encoder inputs on the DMC-30000 motion controller. The termination resistors are 120Ω, and are placed between the positive and negative differential inputs on the Main A, B, Index channels as well as the Auxiliary A and B channels as in Figure A.1.

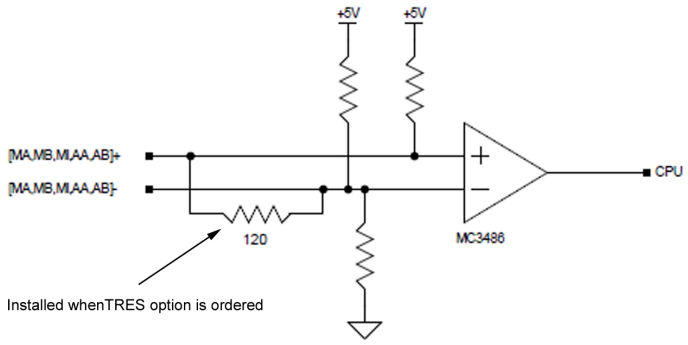


Figure A.1: Encoder Inputs with -TRES option

Single-Ended Encoders: Single-ended encoders will not operate correctly with the termination resistors installed. If a combination of differential encoder inputs with termination resistors and single ended encoders is required on the same controller, contact Galil directly.

DMC-31xxx: When ordered with the DMC-31xxx (Sin/Cos Encoder option), termination resistors will be placed on the Aux Encoder inputs. By default the DMC-31xxx already has termination resistors on the Main Encoder Inputs.

Part number ordering example: DMC-30010-CARD(TRES)

SER – Serial Encoder Interface

The SER enables the DMC-30000 controller to interface to BiSS and SSI encoders. Electrical specifications can be found in the Multi-Function Pins (MF) section of Chapter 3 Connecting Hardware, see the SS and SI commands in the DMC-30000 Command Reference for command information.

Part number ordering example: DMC-30010-CARD(SER)

DMC-31000 – Sin/Cos and 16 bit Analog Inputs

The DMC-31000 provides an interface for 1Vpp Sin/Cos Encoder feedback. See A4 – DMC-31000 for more information.

Part number ordering example: DMC-31011-BOX

Communication Options

RS-422 – Serial Port Serial Communication

The default serial configuration for the DMC-30000 is to have RS-232 communication on the serial port. The controller can be ordered to have RS-422. RS-422 communication is a differentially driven serial communication protocol that should be used when long distance serial communication is required in an application.

RS-422 Serial Port Pinout

Standard connector and cable when DMC-30000 is ordered with RS-422 Option.

Pin	Signal
1	CTS-
2	RXD-
3	TXD-
4	RTS-
5	GND
6	CTS+
7	RXD+
8	TXD+
9	RTS+

JP2 – RS-422 Termination Jumpers

Label	Function (If jumpered)
RXD	RS-422 Option Only: Connects a 120Ohm Termination resistor between the differential "Receive" inputs on the Aux Serial port. Pins 2 and 7 on RS-422 Auxiliary Port.
CTS	RS-422 Option Only: Connects a 120Ohm Termination resistor between the differential "Clear To Send" inputs on the Aux Serial port. Pins 1 and 6 on RS-422 Auxiliary Port.

Part number ordering example: DMC-30010-CARD(422)

Mounting Options

DIN – DIN Rail Mounting

The DIN option on the DMC-30000 motion controller provides DIN rail mounts on the base of the controller. This will allow the controller to be mounted to any standard DIN rail. Requires -BOX option.

Part number ordering example: DMC-30010-BOX-DIN

Internal Amplifier Options

DMC-30012 (DMC-30000 with 800W Sinusoidal Amplifier)

The DMC-30012 (A1 – DMC-30012) provides an amplifier that drives motors operating at 20–80 VDC, up to 10 Amps continuous, 15 Amps peak. The gain settings of the amplifier are user-programmable at 0.4 Amp/Volt, 0.8 Amp/Volt and 1.6 Amp/Volt. The switching frequency is 33 kHz. The amplifier offers protection for over-voltage,

under-voltage, over-current, and short-circuit. The SR90 – SR-49000 Shunt Regulator Option is also available for the DMC-30012.

Part number ordering example: DMC-30012-BOX

DMC-30017 (DMC-30000 with 6Amp stepper driver or 800W Sinusoidal Amplifier)

The DMC-30017 (A3 – DMC-30017) includes a microstepping drive for operating two-phase bipolar stepper motors, the drive can also be configured for a sinusoidally commutated, PWM amplifier for driving brushed or brushless servo motors.

Micro-stepping Drive: The micro-stepping drive produces 256 microsteps per full step or 1024 steps per full cycle which results in 51,200 steps/rev for a standard 200-step motor. The maximum step rate generated by the controller is 3,000,000 microsteps/second. The DMC-30017 can drive stepper motors at up to 6 Amps at 20-80VDC. There are four selectable current gains: 0.75 A, 1.5 A, 3 A and 6A. A selectable low current mode reduces the current by 75% when the motor is not in motion.

Sinusoidally Commutated Amplifier: When set to servo mode, the DMC-30017 has the same specs as the DMC-30012.

Part number ordering example: DMC-30017-BOX

ISCNTL – Isolate Controller Power

The ISCNTL option isolates the power input for the controller from the power input of the amplifiers. With this option, the power is brought in through the 2 pin Molex connector on the side of the controller as shown in the Power Wiring Diagrams for the DMC-30000 section of the Appendix. This option is not valid when Galil amplifies are not ordered with the DMC-30000.

Part number ordering example: DMC-30012-BOX(ISCNTL)

SR90 – SR-49000 Shunt Regulator Option

The SR-49000 is a shunt regulator for the DMC-30000 controller and internal amplifiers. This option is highly recommended for any application where there is a large inertial load, or a gravitational load. The SR-49000 is installed inside the box of the DMC-30000 controller.

The Shunt Regulator activates when the voltage supplied to the amplifier rises above 90V. When activated, the power from the power supply is dissipated through a 5Ω 20W power resistor.

The SR-49000 can be ordered to activate at different voltages. 33V, 66V and 90V are all standard ordering options and can be ordered as -SR33, -SR66 and -SR90 respectively.

Part number ordering example: DMC-30012-BOX-SR90

Miscellaneous Options

RTC – Real Time Clock

The DMC-30000 provides a real time clock feature. The RTC option provides an extended feature set. For details see the Real Time Clock section in Chapter 6.

Real time clock	**DMC-30000**	**DMC-30000(RTC)**
RT providing Hours, Minutes, Seconds	Yes	Yes
RY providing Year, Month of year, Day of month, Day of week	No	Yes
Settable via TIME protocol server (IH and RO commands)	Yes	Yes
Clock persists through DMC power loss	No	Yes
C No-power clock battery life	N/A	1 week

Part number ordering example: DMC-30010-BOX(RTC)

Power Connectors for the DMC-30000

Overview

The DMC-30000 uses different connectors depending upon the type of controller used. The following section details the part numbers used on the controller for the different ordering options. Table A.1: Connector Part Numbers details the connector part numbers used on the DMC-30000 series controllers. Table A.2: Connectors listed by DMC-30000 part number lists the on-board connectors for the different DMC-30000 controller options.

On Board Connector	Common Mating Connectors	Crimp Part Number	Type
Molex# 39-31-0040	Molex# 39-01-2045	Molex# 44476-3112	4 Position
Molex# 39-31-0020	Molex# 39-01-2025	Molex# 44476-3112	2 Position
TE Connectivity# 5-104362-1	Molex# 50-57-9402 Adam-Tech: CDH-02 Oupiin:4077-02HB (1k Min)	Molex# 16-02-0103 Adam-Tech: CDH-C-B (Bulk) Adam-Tech: CDH-C-R (Reel) Oupiin: 4077-PIN-T-T15K (15k Reel)	2 Position
TE Connectivity# 5-104362-3	Molex# 50-57-9404 Adam-Tech: CDH-04 Oupiin: 4077-04HB (1k Min)	Molex# 16-02-0103 Adam-Tech: CDH-C-B (Bulk) Adam-Tech: CDH-C-R (Reel) Oupiin: 4077-PIN-T-T15K (15k Reel)	4 Position

Table A.1: Connector Part Numbers

DMC-30000 Part Number		On Board Connector	Type
DMC-30010	Power	TE Connectivity# 5-104362-3	4 Position
DMC-30011	Power	TE Connectivity# 5-104362-1	2 Position
DMC-30012	Power	Molex# 39-31-0020	2 Position
	Motor	Molex# 39-31-0040	4 Position
DMC-30016	Power	Molex# 39-31-0020	2 Position
	Motor	Molex# 39-31-0040	4 Position
DMC-30017	Power	Molex# 39-31-0020	2 Position
	Motor	Molex# 39-31-0040	4 Position
DMC-30012(ISCNTL)	Power (Controller)	TE Connectivity# 5-104362-1	2 Position
	Power (Amplifier)	Molex# 39-31-0020	2 Position
	Motor	Molex# 39-31-0040	4 Position
DMC-30016(ISCNTL)	Power (Controller)	TE Connectivity# 5-104362-1	2 Position
	Power (Amplifier)	Molex# 39-31-0020	2 Position
	Motor	Molex# 39-31-0040	4 Position
DMC-30017(ISCNTL)	Power (Controller)	TE Connectivity# 5-104362-1	2 Position
	Power (Amplifier)	Molex# 39-31-0020	2 Position
	Motor	Molex# 39-31-0040	4 Position

Table A.2: Connectors listed by DMC-30000 part number

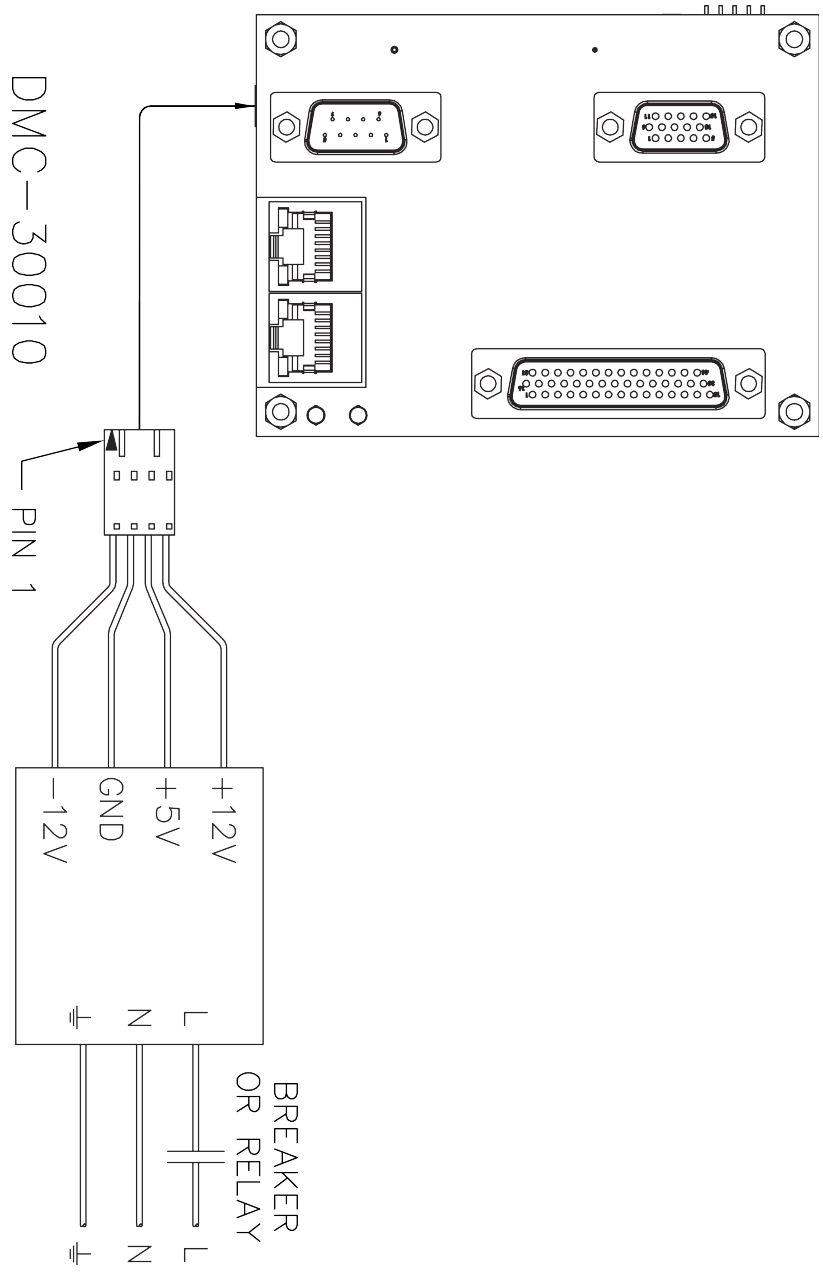
Power Wiring Diagrams for the DMC-30000

The following diagrams shows how to power the different models of the DMC-30000 family. The connectors are keyed and indicate the correct orientation and pin numbers for the power input.

See Input Power Requirements for detailed Electrical Specifications.

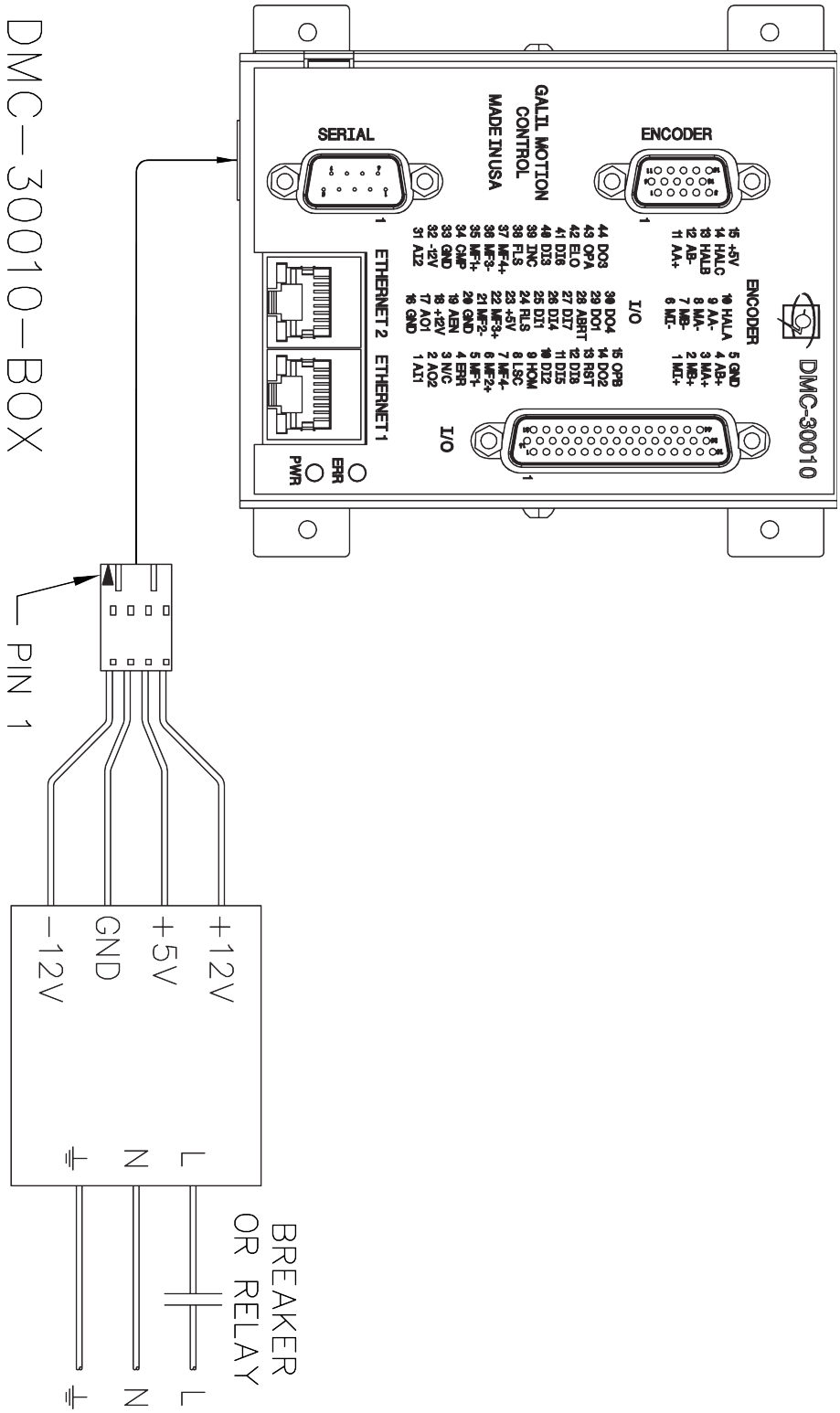
DMC-30010-CARD

Requires a +5VDC and +/-12VDC triple power supply.



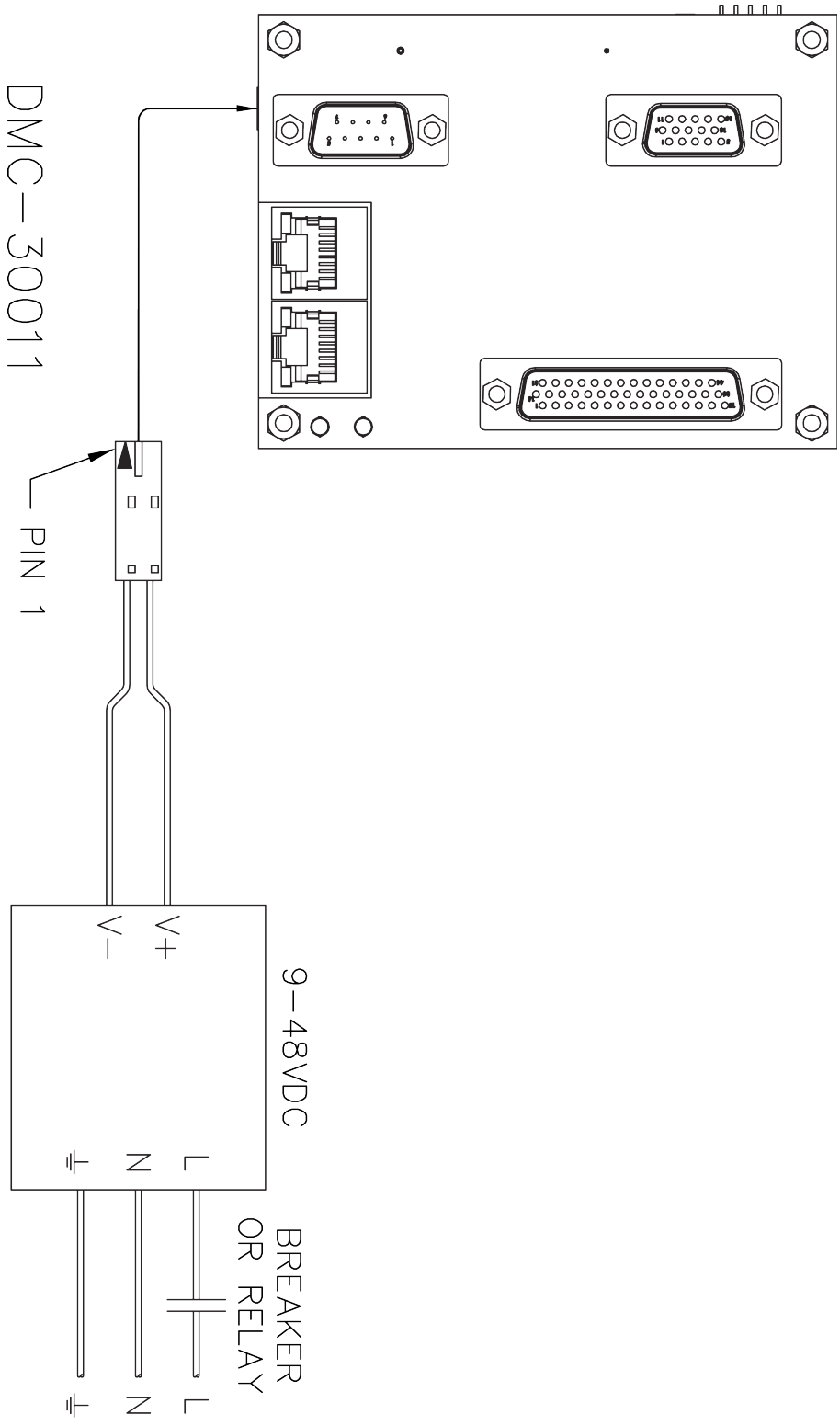
DMC-30010-BOX

Requires a +5VDC and +/-12VDC triple power supply.



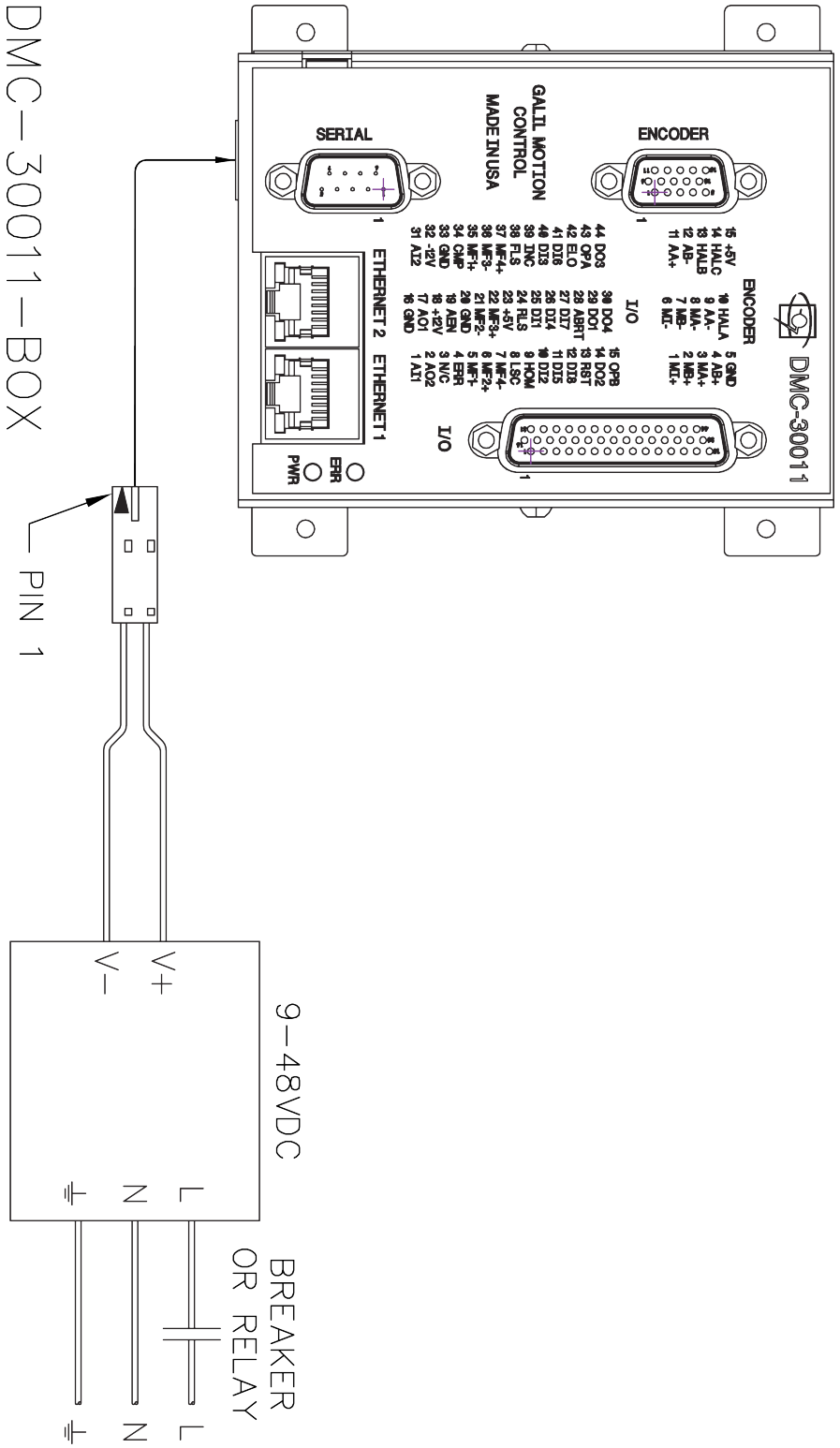
DMC-30011-CARD

Requires a +9VDC to +48VDC power supply.



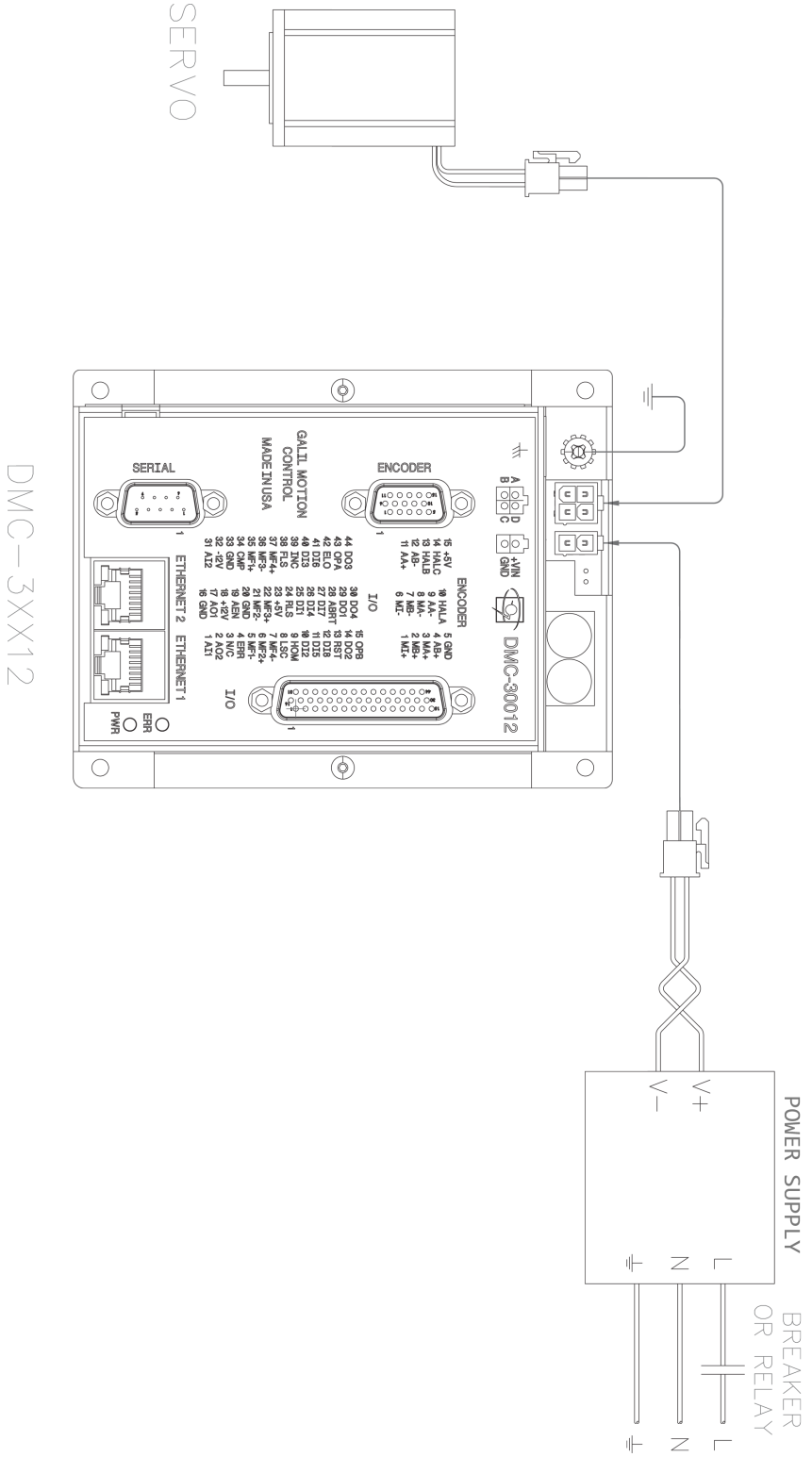
DMC-30011-BOX

Requires a +9VDC to +48VDC power supply.



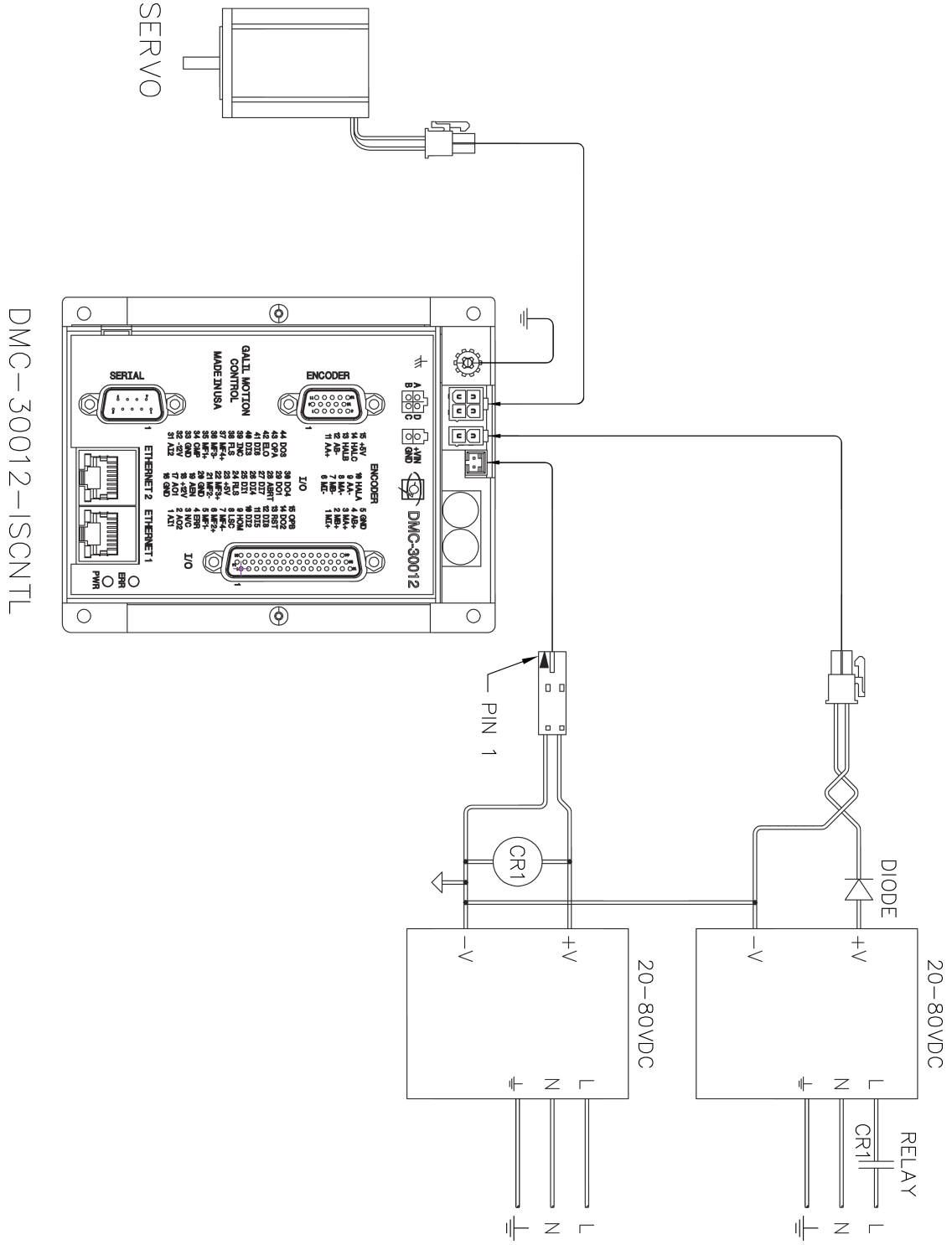
DMC-30012-BOX, DMC-30016-BOX and DMC-30017-BOX

See power requirements for individual configurations in the Appendices.



DMC-30012-BOX(ISCNTL), DMC-30016-BOX(ISCNTL) and DMC-30017-BOX(ISCNTL)

Requires two DC power supplies, see power requirements for individual configurations in the Appendices.



Connectors for DMC-30000 (Pin-outs)

J5 - I/O 44 pin HD D-Sub Connector (Female)

Pin #	Label	Description	Pin #	Label	Description	Pin #	Label	Description
1	AI1	Analog Input 1	16	AGND	Analog Ground	31	AI2	Analog Input 2
2	AO2	Analog Output 2	17	AO1	MCMD/Analog Output 1[1]	32	-12V	-12V
3	N/C	No Connect	18	+12V	+12V	33	GND	Ground
4	ERR	Error Output	19	AEN	Amplifier Enable	34	CMP	Output Compare
5	MF1 -	Multi-Function 1 -	20	GND	Ground	35	MF1 +	Multi-Function 1 +
6	MF2 +	Multi-Function 2 +	21	MF2 -	Multi-Function 2 -	36	MF3 -	Multi-Function 3 -
7	MF4 -	Multi-Function 4 -	22	MF3 +	Multi-Function 3 +	37	MF4 +	Multi-Function 4+
8	LSC	Limit Switch Common	23	+5V	+5V	38	FLS	Forward Limit Switch
9	HOM	Home Switch Input	24	RLS	Reverse Limit	39	INC	Input Common
10	DI2	Digital Input 2	25	DI1	Digital Input 1	40	DI3	Digital Input 3
11	DI5	Digital Input 5	26	DI4	Digital Input 4	41	DI6	Digital Input 6
12	DI8	Digital Input 8	27	DI7	Digital Input 7	42	ELO	Electronic Lockout
13	RST	Reset Input	28	ABRT	Abort Input	43	OPA	Output GND/PWR (Bank 0)
14	DO2	Digital Output 2	29	DO1	Digital Output 1	44	DO3	Digital Output 3
15	OPB	Output PWR/GND (Bank 0)	30	DO4	Digital Output 4			

[1]AO1, Analog Output 1 is used as the motor command output for the DMC-3xx10 and DMC-3xx11.

Multi-Functional Pins (MFn+/-)

The Multi-Functional Pins on the DMC-30000 have different functionalities dependent upon how the controller was ordered and how the controller is setup by the user. If the controller is ordered with -SER (serial encoder interface), then the MF pins can be used to interface to a serial encoder. MF1 and MF2 are only used for the Main serial encoder input, MF2 and MF3 are used for the Aux serial encoder input. See the SI and SS commands in the command reference for more detail.

When the controller is setup for stepper motor operation, the MF 2 and MF4 pins are used for step and direction respectively.

Single Description for Multi-Functional Pins			
Label	**Pin #**	**MT +/-2 or +/-2.5**	**-SER option with BiSS or SSI Enabled**
MF1 +	35	No Connect	Main Axis Data + (D0+ or SLO+)
MF1 -	5	No Connect	Main Axis Data - (D0- or SLO-)
MF2 +	6	STEP +	Main Axis Clock + (C0+ or MA+)
MF2 -	21	STEP -	Main Axis Clock - (C0- or MA-)
MF3 +	22	No Connect	Aux Axis Data + (D1+ or SLO+)
MF3 -	36	No Connect	Aux Axis Data - (D1- or SLO-)
MF4 +	37	DIR +	Aux Axis Clock + (C1+ or MA+)
MF4 -	7	DIR -	Aux Axis Clock - (C1- or MA-)

J4 - Encoder 15 pin HD D-Sub Connector (Female)

Pin #	Label	Description
1	MI+	I+ Index Pulse Input
2	MB+	B+ Main Encoder Input
3	MA+	A+ Main Encoder Input
4	AB+	B+ Aux Encoder Input
5	GND	Digital Ground
6	MI-	I- Index Pulse Input
7	MB-	B- Main Encoder Input
8	MA-	A- Main Encoder Input
9	AA-	A- Aux Encoder Input
10	HALA	A Channel Hall Sensor
11	AA+	A+ Aux Encoder Input
12	AB-	B- Aux Encoder Input
13	HALB	B Channel Hall Sensor
14	HALC	C Channel Hall Sensor
15	+5V	+5V

J1/J2 – Ethernet (RJ45)

Pin #	Signal
1	TXP
2	TXN
3	RXP
4	NC
5	NC
6	RXN
7	NC
8	NC

The Ethernet connection is Auto MDIX, 100bT/10bT.

J3 - RS-232-Main Port (Male)

Standard connector and cable, 9Pin

Pin	Signal
1	NC
2	TXD
3	RXD
4	NC
5	GND
6	NC
7	CTS
8	RTS
9	NC

JP1 - Jumper Description for DMC-30000

Label	Function (If jumpered)
OPT	Reserved
MO	When controller is powered on or reset, Amplifier Enable lines will be in a Motor Off state. A SH will be required to re-enable the motors.
19.2	Baud Rate setting – see table below
UG	Used to upgrade controller firmware when resident firmware is corrupt.
MR	Master Reset enable. Returns controller to factory default settings and erases FLASH. Requires power-on or RESET to be activated.

Baud Rate Jumper Settings

19.2	BAUD RATE
ON	19200
OFF	115200 (Recommended)

Signal Descriptions for DMC-30000

Outputs

Analog Outputs 1,2 / Motor Command	+/- 10 Volt range signal for driving amplifier or for a general purpose analog output. In servo mode, motor command output is updated at the controller sample rate. In the motor off mode, this output is held at the OF command level.
Amplifier Enable	Signal to disable and enable an amplifier. Amp Enable goes low on Abort and OE1. 5V HAEN only
MF2 – Step Output	For stepper motors: When MT is set to 2,-2,2.5 or -2.5 the MF2 pins produces a series of pulses for input to a step motor driver. The pulses may either be low or high. The pulse width is 50%. With an internal amplifier, BR-1 must be set as well as MT.
MF4 - Direction	For stepper motors: When MT is set to 2,-2,2.5 or -2.5 the MF2 pins produces the direction output for stepper motors. With an internal amplifier, BR-1 must be set as well as MT.
Error	The signal goes low when the position error on any axis exceeds the value specified by the error limit command, ER.
Output 1-Output 4	The optically isolated outputs are uncommitted and may be designated by the user to trigger external events. The output lines are toggled by Set Bit, SB, and Clear Bit, CB, instructions. The OP instruction is used to define the state of all the bits of the Output port.

Inputs

Encoder, MA+, MB+	Position feedback from incremental encoder with two channels in quadrature, CHA and CHB. The encoder may be analog or TTL. Any resolution encoder may be used as long as the maximum frequency does not exceed 15,000,000 quadrature states/sec. The controller performs quadrature decoding of the encoder signals resulting in a resolution of quadrature counts (4 x encoder cycles). Note: Encoders that produce outputs in the format of pulses and direction may also be used by inputting the pulses into CHA and direction into Channel B and using the CE command to configure this mode.
Encoder Index, MI+	Once-Per-Revolution encoder pulse. Used in Homing sequence or Find Index command to define home on an encoder index.
Encoder, MA-, MB-, MI-	Differential inputs from encoder. May be input along with CHA, CHB for noise immunity of encoder signals. The CHA- and CHB- inputs are optional.
Auxiliary Encoder, AA+, AB+, Aux A-, Aux B-	Inputs for additional encoder. Used when an encoder on both the motor and the load is required. Not available on axes configured for step motors.
Abort	A low input stops commanded motion instantly without a controlled deceleration. Also aborts motion program.

Reset	A low input resets the state of the processor to its power-on condition. The previously saved state of the controller, along with parameter values, and saved sequences are restored.
Electronic Lock Out	Controllers with Internal Amplifiers Only. Input that when triggered will shut down the amplifiers at a hardware level. Useful for safety applications where amplifiers must be shut down at a hardware level.
Forward Limit Switch	When active, inhibits motion in forward direction. Also causes execution of limit switch subroutine, #LIMSWI. The polarity of the limit switch may be set with the CN command.
Reverse Limit Switch	When active, inhibits motion in reverse direction. Also causes execution of limit switch subroutine, #LIMSWI. The polarity of the limit switch may be set with the CN command.
Home Switch	Input for Homing (HM) and Find Edge (FE) instructions. Upon BG following HM or FE, the motor accelerates to slew speed. A transition on this input will cause the motor to decelerate to a stop. The polarity of the Home Switch may be set with the CN command.
Input 1 - Input 8	Uncommitted inputs. May be defined by the user to trigger events. Inputs are checked with the Conditional Jump instruction and After Input instruction or Input Interrupt. Input 1 is latch A if the high speed position latch function is enabled.
Latch	High speed position latch to capture axis position on occurrence of latch signal. AL command arms latch. Input 1 is latch A.

List of Other Publications

"Step by Step Design of Motion Control Systems"

by Dr. Jacob Tal

"Motion Control Applications"

by Dr. Jacob Tal

"Motion Control by Microprocessors"

by Dr. Jacob Tal

Training Seminars

Galil, a leader in motion control with over 500,000 controllers working worldwide, has a proud reputation for anticipating and setting the trends in motion control. Galil understands your need to keep abreast with these trends in order to remain resourceful and competitive. Through a series of seminars and workshops held over the past 20 years, Galil has actively shared their market insights in a no-nonsense way for a world of engineers on the move. In fact, over 10,000 engineers have attended Galil seminars. The tradition continues with three different seminars, each designed for your particular skill set-from beginner to the most advanced.

MOTION CONTROL MADE EASY

WHO SHOULD ATTEND

Those who need a basic introduction or refresher on how to successfully implement servo motion control systems.

TIME: 4 hours (8:30 am-12:30 pm)

ADVANCED MOTION CONTROL

WHO SHOULD ATTEND

Those who consider themselves a "servo specialist" and require an in-depth knowledge of motion control systems to ensure outstanding controller performance. Also, prior completion of “Motion Control Made Easy" or equivalent is required. Analysis and design tools as well as several design examples will be provided.

TIME: 8 hours (8:00 am-5:00 pm)

PRODUCT WORKSHOP

WHO SHOULD ATTEND

Current users of Galil motion controllers. Conducted at Galil’s headquarters in Rocklin, CA, students will gain detailed understanding about connecting systems elements, system tuning and motion programming. This is a “hands-on” seminar and students can test their application on actual hardware and review it with Galil specialists.

Attendees must have a current application and recently purchased a Galil controller to attend this course.

TIME: Two days (8:30-4:30pm)

http://www.galilmc.com/learning/training-at-galil.php

Contacting Us

Galil Motion Control

270 Technology Way

Rocklin, CA 95765

Phone: 916-626-0101

Fax: 916-626-0102

E-Mail Address: support@galilmc.com

Web: http://www.galilmc.com/

WARRANTY

All controllers manufactured by Galil Motion Control are warranted against defects in materials and workmanship for a period of 18 months after shipment. Motors, and Power supplies are warranted for 1 year. Extended warranties are available.

In the event of any defects in materials or workmanship, Galil Motion Control will, at its sole option, repair or replace the defective product covered by this warranty without charge. To obtain warranty service, the defective product must be returned within 30 days of the expiration of the applicable warranty period to Galil Motion Control, properly packaged and with transportation and insurance prepaid. We will reship at our expense only to destinations in the United States and for products within warranty.

Call Galil to receive a Return Materials Authorization (RMA) number prior to returning product to Galil.

Any defect in materials or workmanship determined by Galil Motion Control to be attributable to customer alteration, modification, negligence or misuse is not covered by this warranty.

EXCEPT AS SET FORTH ABOVE, GALIL MOTION CONTROL WILL MAKE NO WARRANTIES EITHER EXPRESSED OR IMPLIED, WITH RESPECT TO SUCH PRODUCTS, AND SHALL NOT BE LIABLE OR RESPONSIBLE FOR ANY INCIDENTAL OR CONSEQUENTIAL DAMAGES.

A1 – DMC-30012

Description

The DMC-30012 includes a sinusoidally commutated, PWM amplifier for driving 3 phase brushless servo motors or a brushed motor. Each amplifier drives motors operating at up to 10 Amps continuous, 15 Amps peak, 20–80 VDC. The gain settings of the amplifier are user-programmable at 0.4 Amp/Volt, 0.8 Amp/Volt and 1.6 Amp/Volt. The switching frequency is 33 kHz. The amplifier offers protection for over-voltage, under-voltage, over-current, short-circuit and over-temperature. A shunt regulator option is available. If higher voltages are required, please contact Galil.

If the application has a potential for regenerative energy it is recommended to order the controller with the ISCNTL – Isolate Controller Power option and the SR90 – SR-49000 Shunt Regulator Option.

Note: Do not "hot swap" the motor power or supply voltage power input connections. If the amp is enabled when the motor connector is connected or disconnected, damage to the amplifier can occur. Galil recommends powering the controller and amplifier down before changing the connector, and breaking the AC side of the power supply connection in order to power down the amplifier. The ELO input may be used to cut power to the motors in an Emergency Stop or Abort situation.

Figure A1.1: DMC-30012

Electrical Specifications

The amplifier is a brush/brushless transconductance PWM amplifier. The amplifier operates in torque mode, and will output a motor current proportional to the command signal input.

Supply Voltage:	20-80 VDC
Continuous Current:	10 Amps
Peak Current:	15 Amps
Nominal Amplifier Gain:	0.8 Amps/Volt
Switching Frequency:	33 kHz
Minimum Inductance:	Vsupply = 24VDC – 0.75 mH
	Vsupply = 48VDC – 1.2 mH
Brushless Motor Commutation angle:	120°

Mating Connectors

	On Board Connector	Terminal Pins
POWER	2-pin Molex Mini-Fit, Jr.™ MOLEX# 39-31-0020	MOLEX#44476-3112
A,B,C,D: 4-pin Motor Power Connectors	4-pin Molex Mini-Fit, Jr.™ MOLEX# 39-31-0040	MOLEX#44476-3112

For mating connectors see http://www.molex.com/

Motor Connector

Power Connector

Power Connector	
Pin Number	Connection
1	DC Power Supply Ground
2	+VS (DC Power)
Motor Connector	
1	Phase C
2	Phase B (N/C for Bushed Motors)
3	No Connect
4	Phase A

Operation

Setting up the Brushless Mode and finding proper commutation

The 6 commands used for set up are the BA, BM, BX, BZ, BC and BI commands. Please see the command reference for details.

For detailed information on setting up commutation on the DMC-30012 can be found here:

http://www.galilmc.com/techtalk/drives/wiring-a-brushless-motor-for-galils-sine-amplifier/

1. Issue the BA command to specify which axis you want to use the sinusoidal amplifier on
2. Calculate the number of encoder counts per magnetic cycle. For example, in a rotary motor that has 2 pole pairs and 10,000 counts per revolution, the number of encoder counts per magnetic cycle would be 10,000/2 = 5000. Assign this value to BM
3. Issue either the BZ or BX command. Either the BX or BZ command must be executed on every reset or power-up of the controller.

- BZ Command:

  Issue the BZ command to lock the motor into a phase. Note that this will cause up to ½ a magnetic cycle of motion. Be sure to use a high enough value with BZ to ensure the motor is locked into phase properly.

- BX Command:

  Issue the BX command. The BX command utilizes a minimal movement algorithm in order to determine the correct commutation of the motor.

Setting Amplifier Gain and Current Loop Gain

The AG command will set the amplifier gain (Amps/Volt), and the AU command will set the current loop gain for the DMC-30012. The current loop gain will need to be set based upon the bus voltage and inductance of the motor and is critical in providing the best possible performance of the system.

AG command:

The DMC-30012 has 3 amplifier gain settings. The gain is set with the AG command as shown in Table A1.1 for AG n=m:

AG setting	Gain Value
m = 0	0.4 A/V
m = 1	0.8 A/V
m = 2	1.6 A/V

Table A1.1: Amplifier Gain Settings for DMC-30012

The axis must be in a motor off (MO) state prior to execution of the AG command. With an amplifier gain of 2 (1.6A/V) the maximum motor command output is limited to 5V (TL of 5).

AU command:

Proper configuration of the AU command is essential to optimum operation of the DMC-30012. This command sets the gain for the current loop on the amplifier. The goal is to set the gain as high as possible without causing the current loop to go unstable. In most cases AU 0 should not be used. Table A1.2 indicates the recommended AUn=m settings for 24 and 48 VDC power supplies.

To set the AU command, put the axis in a motor off (MO) state, set the preferred AG setting, and then set the AU setting. To verify that the current loop is stable, set the PID's to 0 (KP, KD and KI) and then enable the axis (SH). An unstable current loop will result in oscillations of the motor or a high frequency “buzz” from the motor.

Vsupply VDC	Inductance L (mH)	m =
24	-	0
24	L < 1	1
24	1 < L < 2.3	2
24	2.3 < L < 4.2	3
24	4.2 < L	4
48	-	0
48	L < 2.4	1
48	2.4 < L < 4.2	2
48	4.2 < L < 7	3
48	7 < L	4

Table A1.2: Amplifier Current Loop Gain Settings

Setting Peak and Continuous Current (TL and TK)

To set TL and TK for a particular motor, find the continuous current and peak current ratings for that motor and divide that number by the amplifier gain. For example, a particular motor has a continuous current rating of 2.0 A and peak current rating of 5.0 A. With an AG setting of 1, the amplifier gain of the DMC-30012 is 0.8A/V

TL setting = (2.0A) / (0.8A/V) = 2.5V (TL n=2.5)

TK setting = (5.0A) / (0.8A/V) = 7.5V (TK n=6.25)

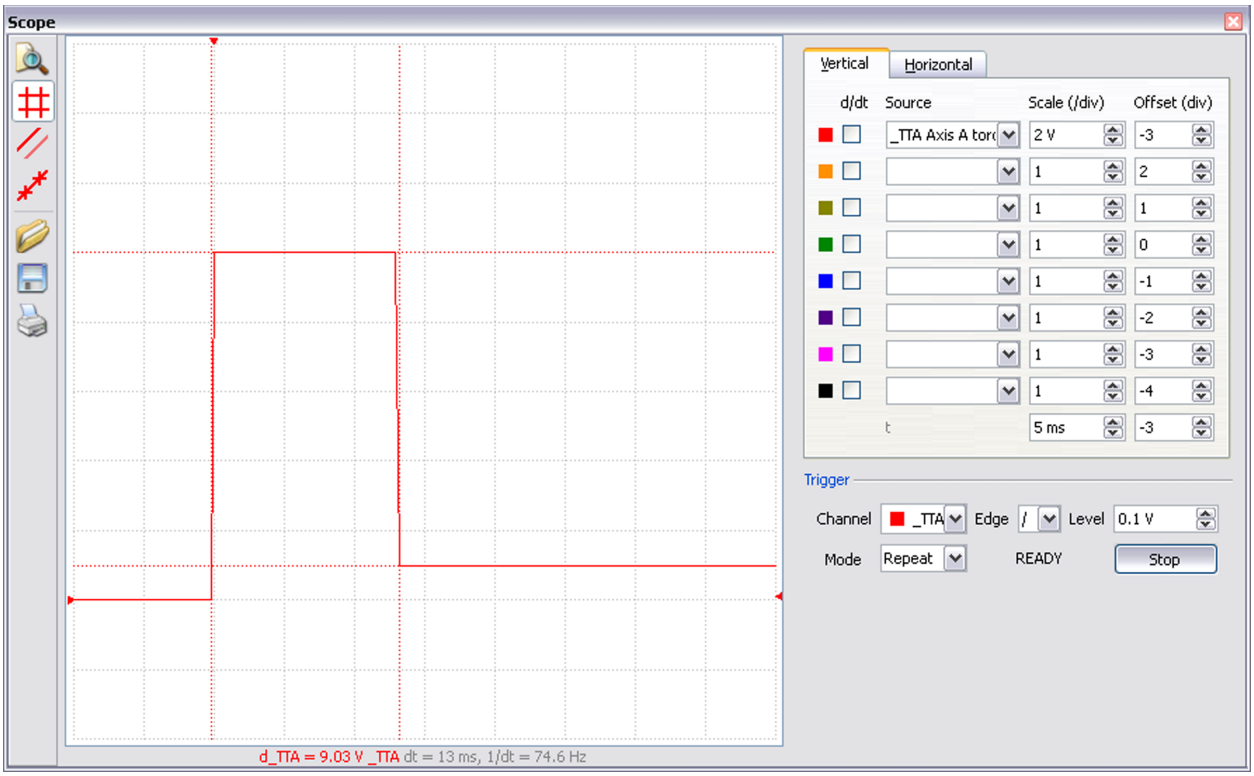


Figure A1.2: Peak Current Operation

Brushed Motor Operation

The AMP-43540 can be setup to run brushed motors by setting the BR command to 1 for a particular axis. Wire the motor power leads to phases A and C on the motor power connector. Do not set BA, BM or use the BX command for any axis that is driving a brushed motor.

ELO Input

If the ELO input on the controller is triggered, the amplifier will be shut down at a hardware level, the motors will be essentially in a Motor Off (MO) state. TA3 will return a 3 and the #AMPERR routine will run when the ELO input is triggered. To recover from an ELO, an MO then SH must be issued, or the controller must be reset.

It is recommended that OE1 be used for all axes when the ELO is used in an application.

Error Monitoring and Protection

The amplifier is protected against over-voltage, under-voltage, over-temperature, and over-current for brush and brushless operation. The controller will monitor the error conditions and respond as programmed in the application. The errors are monitored via the TA command. TA n may be used to monitor the errors with n = 0, 2, or 3. The command will return an eight bit number representing specific conditions. TA0 will return errors with regard to under voltage, over voltage, over current, and over temperature. TA2 will monitor if the amplifier current exceeds the continuous setting, and TA3 will return if the ELO input has been triggered.

The user also has the option to include the special label #AMPERR in their program to handle amplifier errors. As long as a program is executing in thread zero and the #AMPERR label is included, when an error is detected the program will jump to the label and execute the user defined routine. Note that the TA command is a monitoring function only, and does not generate an error condition.

See the TA command for detailed information on bit status during error conditions.

Under-Voltage Protection

If the supply to the amplifier drops below 18 VDC, the amplifier will be disabled. The amplifier will return to normal operation once the supply is raised above the 18V threshold.

NOTE: If there is an #AMPERR routine and the controller is powered before the amplifier, then the #AMPERR routine will automatically be triggered.

Over-Voltage Protection

If the voltage supply to the amplifier rises above 94 VDC, then the amplifier will automatically disable. The amplifier will re-enable when the supply drops below 90 V.

The over voltage condition will not permanently shut down the amplifier or trigger the #AMPERR routine. The amplifier will be momentarily disabled; when the condition goes away, the amplifier will continue normal operation assuming it did not cause the position error to exceed the error limit.

Over-Current Protection

The amplifier also has circuitry to protect against over-current. If the total current from a set of 2 axes (ie A and B or C and D) exceeds 20 A, the amplifier will be disabled. The amplifier will not be re-enabled until there is no longer an over-current draw and then either SH command has been sent or the controller is reset. Since the DMC-30012 is a trans-conductance amplifier, the amplifier will never go into this mode during normal operation. The amplifier will be shut down regardless of the setting of OE, or the presence of the #AMPERR routine.

NOTE: If this fault occurs, it is indicative of a problem at the system level. An over-current fault is usually due to a short across the motor leads or a short from a motor lead to ground.

Over-Temperature Protection

The amplifier is also equipped with over-temperature protection.

If the average heat sink temperature rises above 80°C, then the amplifier will be disabled. The over-temperature condition will trigger the #AMPERR routine if included in the program on the controller.

The amplifier will not be re-enabled until the temperature drops below 80°C and then either an SH command is sent to the controller, or the controller is reset (RS command or power cycle).

A2 – DMC-30016

Description

The DMC-30016 contains a drive for operating two-phase bipolar step motors. The DMC-30016 requires a single 12-30 VDC input. The unit is user-configurable for 0.5 to 1.4 Amps per phase and for full-step, half-step, 1/4 step or 1/16 step.

Note: Do not "hot swap" the motor power or supply voltage power input connections. If the amp is enabled when the motor connector is connected or disconnected, damage to the amplifier can occur. Galil recommends powering the controller and amplifier down before changing the connector, and breaking the AC side of the power supply connection in order to power down the amplifier. The ELO input may be used to cut power to the motors in an Emergency Stop or Abort situation.

Figure A2.1: DMC-30016

Electrical Specifications

DC Supply Voltage:	12-30 VDC In order to run the DMC-30016 in the range of 12-20 VDC, the ISCNTL – Isolate Controller Power option must be ordered
Max Current (per axis)	1.4 Amps/Phase Amps (Selectable with AG command)
Maximum Step Frequency:	3 MHz
Motor Type:	Bipolar 2 Phase

Mating Connectors

	On Board Connector	Terminal Pins
POWER	6-pin Molex Mini-Fit, Jr.™ MOLEX# 39-31-0060	MOLEX#44476-3112
A,B,C,D: 4-pin Motor Power Connectors	4-pin Molex Mini-Fit, Jr.™ MOLEX# 39-31-0040	MOLEX#44476-3112

For mating connectors see http://www.molex.com/

Motor Connector

Power Connector

Power Connector	
Pin Number	Connection
1	DC Power Supply Ground
2	+VS (DC Power)
Motor Connector	
1	B-
2	B+
3	A-
4	A+

Note: The stepper motor wiring on the DMC-30016 is not compatible with other Galil stepper drivers such as the SDM-44140 and SDM-44040.

Operation

The AG command sets the current on each axis, the LC command configures each axis's behavior when holding position and the YA command sets the step driver resolution. These commands are detailed below, see also the command reference for more information:

Stepper Mode

With the DMC-30016, the controller will default to MT-2 (stepper motor). To set the controller for external servo mode, set MT1.

The DMC-30016 should be setup for Active High step pulses (MT-2 or MT-2.5).

Current Level Setup (AG Command)

AG configures how much current the DMC-30016 delivers to each motor. It is settable in ~7mA increments from 0.5 to 1.4 Amps

Low Current Mode (LC):

LC configures the behavior when holding position (when RP is constant). The settings are shown in Table A2.1 for LC m.

LC Setting	Mode	Description
m = 0	Full Current	Motor uses 100% of current at all times when enabled
m = 1	Low Current	Motor uses 25% of current while at resting state
m = 2 - 32767	Delayed Low Current	'm' specifies the number of samples to wait between the end of the move and when the current is cut to 25%

Table A2.1: LC settings for DMC-30016

Step Drive Resolution Setting (YA command)

When using the DMC-30016, the step drive resolution can be set with the YA command as shown in Table A2.2 for YA m.

YA setting	Step Resolution
m = 1	Full (70% holding current)
m = 2	Half
m = 4	1/4
m = 16	1/16

Table A2.2: YA settings

ELO Input

If the ELO input on the controller is triggered, the amplifier will be shut down at a hardware level, the motors will be essentially in a Motor Off (MO) state. TA3 will return a 3 and the #AMPERR routine will run when the ELO input is triggered. To recover from an ELO, an MO then SH must be issued, or the controller must be reset.

It is recommended that OE1 be used for all axes when the ELO is used in an application.

See the ELO (Electronic Lock-Out) Input section in Chapter 3 Connecting Hardware for information on connecting the ELO input.

Using External Amplifiers

Use the connectors on top of the controller to access necessary signals to run external amplifiers. For more information on connecting external amplifiers, see Connecting to External Amplifiers in Chapter 2.

Protection Circuitry

The DMC-30016 has short circuit protection. The short circuit protection will protect against phase to phase shorts, a shorted load and a short to ground or chassis.

In the event of any of a fault, bit 0 of TA0 will be set DMC-30016 will be disabled.

In the event that power is removed to the DMC-30016 but not to the controller, an amplifier error will occur.

To recover from an error state, the controller must be set into MO state, LC must set to 0 and then the SH command must be issued.

A3 – DMC-30017

Description

The DMC-30017 includes a microstepping drive for operating two-phase bipolar stepper motors, the drive can also be configured for a sinusoidally commutated, PWM amplifier for driving 3 phase brushless servo motors or a brushed motor.

Micro-stepping Drive:

The micro-stepping drive produces 256 microsteps per full step or 1024 steps per full cycle which results in 51,200 steps/rev for a standard 200-step motor. The maximum step rate generated by the controller is 3,000,000 microsteps/second. The DMC-30017 can drive stepper motors at up to 6 Amps at 20-80VDC. There are four selectable current gains: 0.75 A, 1.5 A, 3 A and 6A. A selectable low current mode reduces the current by 75% when the motor is not in motion.

Sinusoidally Commutated Amplifier:

The DMC-30017 can also be used as a sinusoidally commutated amplifier. See A1 – DMC-30012 for specifications. To get the DMC-30017 into this mode, issue MT 1.

Note: Do not "hot swap" the motor power or supply voltage power input connections. If the amp is enabled when the motor connector is connected or disconnected, damage to the amplifier can occur. Galil recommends powering the controller and amplifier down before changing the connector, and breaking the AC side of the power supply connection in order to power down the amplifier. The ELO input may be used to cut power to the motors in an Emergency Stop or Abort situation.

Electrical Specifications

Supply Voltage:	20-80 VDC
Maximum Current:	6.0 Amps
Maximum Step Frequency:	3 MHz
Step Resolution:	256 steps/full step
Switching Frequency:	33 kHz
Minimum Inductance:	Vsupply = 24VDC – 0.75 mH
	Vsupply = 48VDC – 1.2 mH

Mating Connectors

	On Board Connector	Terminal Pins
POWER	2-pin Molex Mini-Fit, Jr.™ MOLEX# 39-31-0020	MOLEX#44476-3112
A,B,C,D: 4-pin Motor Power Connectors	4-pin Molex Mini-Fit, Jr.™ MOLEX# 39-31-0040	MOLEX#44476-3112

For mating connectors see http://www.molex.com/

Motor Connector

Power Connector

Power Connector	
Pin Number	Connection
1	DC Power Supply Ground
2	+VS (DC Power)
Motor Connector	
1	B-
2	B+
3	A-
4	A+

Note: The stepper motor wiring on the DMC-30017 is not compatible with other Galil stepper drivers such as the SDM-44140 and SDM-44040.

Operation

Stepper Mode

With the DMC-30017, the controller will default to MT-2 (stepper motor). To set the controller for servo mode, set MT1. See A1 – DMC-30012 for further information regarding running in servo mode.

Setting the Current (AG):

The DMC-30017 has 4 amplifier gain (current) settings. The gain is set with the AG command as shown in Table A3.1 for AG m:

AG setting	Gain Value
m = 0	0.75 A/Phase
m = 1	1.5 A/Phase
m = 2	3 A/Phase
m = 3	6 A/Phase

Table A3.1: Amplifier Gain Settings for DMC-30017

The axis must be in a motor off (MO) state prior to execution of the AG command.

The current ratings are peak current per phase.

Low Current Mode (LC):

LC configures the behavior when holding position (when RP is constant). The settings are shown in Table A3.2 for LC m.

LC Setting	Mode	Description
m = 0	Full Current	Motor uses 100% of current at all times when enabled
m = 1	Low Current	Motor uses 25% of current while at resting state
m = 2 - 32767	Delayed Low Current	'm' specifies the number of samples to wait between the end of the move and when the current is cut to 25%

Table A3.2: LC settings for DMC-30017

ELO Input

If the ELO input on the controller is triggered, the amplifier will be shut down at a hardware level, the motors will be essentially in a Motor Off (MO) state. TA3 will return a 3 and the #AMPERR routine will run when the ELO input is triggered. To recover from an ELO, an MO then SH must be issued, or the controller must be reset.

It is recommended that OE1 be used for all axes when the ELO is used in an application.

Error Monitoring and Protection

The amplifier is protected against over-voltage, under-voltage, over-temperature, and over-current for brush and brushless operation. The controller will monitor the error conditions and respond as programmed in the application. The errors are monitored via the TA command. TA n may be used to monitor the errors with n = 0, 2, or 3. The command will return an eight bit number representing specific conditions. TA0 will return errors with regard to under voltage, over voltage, over current, and over temperature. TA2 will monitor if the amplifier current exceeds the continuous setting, and TA3 will return if the ELO input has been triggered.

The user also has the option to include the special label #AMPERR in their program to handle amplifier errors. As long as a program is executing in thread zero and the #AMPERR label is included, when an error is detected the program will jump to the label and execute the user defined routine. Note that the TA command is a monitoring function only, and does not generate an error condition.

See the TA command for detailed information on bit status during error conditions.

See the the DMC-30012 Error Monitoring and Protection section for information regarding functionality of the specific types of protection on the DMC-30017.

A4 – DMC-31000

Description

The DMC-31000 is an option that allows for the controller to accept sinusoidal encoder signals instead of digital encoder signals. The DMC-31000 provides interpolation of up to four 1-volt differential sinusoidal encoders resulting in a higher position resolution. The AFn command selects sinusoidal interpolation where n specifies 2^n interpolation counts per encoder cycle (n=5 to 12). For example, if the encoder cycle is 40 microns, AF10 results in 2^{10}=1024 counts per cycle, or a resolution of 39 nanometers per count. With the DMC-31000, the sinusoidal encoder inputs replace the main digital encoder inputs.

Any DMC-31000 will be installed with firmware specific for the implementation of Sin/Cos encoders. With this firmware the maximum speed settings will be increased from 22,000,000 to 50,000,000 counts/second.

Note: The encoder must be wired to the machine prior to issuing the AF command.

Analog Inputs

With the DMC-31000 the analog inputs upgraded to 16 bit +/-10V configurable analog inputs, see the Analog Inputs section in Chapter 3 Connecting Hardware for more information.

DMC-31000 Encoder 15 pin HD D-Sub Connector (Female)

Pin #	Label	Description
1	MI+	Index Pulse Input - V_0+ (Sin/Cos) or I+ (Digital)
2	MB+	Main Encoder Input - V_2+ (Sin/Cos) or B+ (Digital)
3	MA+	Main Encoder Input - V_1+ (Sin/Cos) or A+ (Digital)
4	AB+	Aux Encoder Input- B+ (Digital)
5	GND	Digital Ground
6	MI-	Index Pulse Input- V_0- (Sin/Cos) or I- (Digital)
7	MB-	Main Encoder Input- V_2- (Sin/Cos) or B- (Digital)
8	MA-	Main Encoder Input- V_1- (Sin/Cos) or A- (Digital)
9	AA-	Aux Encoder Input- A- (Digital)
10	HALA	A Channel Hall Sensor
11	AA+	Aux Encoder Input- A+ (Digital)
12	AB-	Aux Encoder Input- B- (Digital)
13	HALB	B Channel Hall Sensor
14	HALC	C Channel Hall Sensor

15	+5V	+5V

Theory of Operation

Traditional quadrature rotary encoders work by having two sets of lines inscribed radially around the circumference of an optical disk. A light is passed through each of these two sets of lines. On the other side of the gratings, photo sensors detect the presence (or absence) of these lines. These two sets of lines are offset from each other such that one leads the other by one quarter of a complete cycle as shown in Figure A4.1 below. These signals are commonly referred to as the Channels A and B. The direction of rotation of the encoder can be inferred by which of the A and B signals leads the other. Each rising or falling edge indicates one quadrature count. Thus for a complete cycle of the square wave there are a total of four encoder counts.

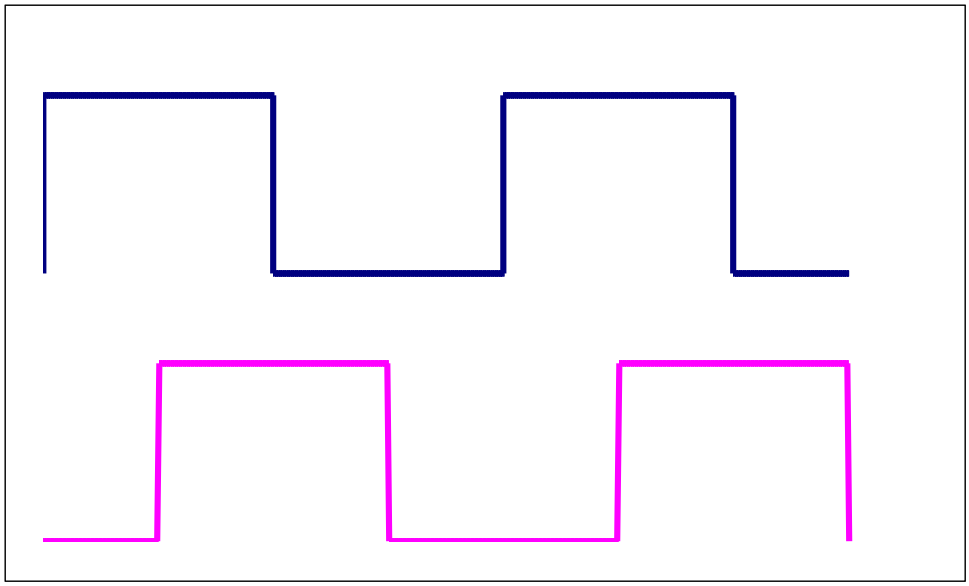


Figure A4.1: Quadrature Encoder Signals

A sinusoidal encoder is similar to a quadrature encoder in that it produces two signals that are read from two sets of lines inscribed on an optical disk. The difference is that the two signals are output as analog sinusoidal waves as shown in Figure A4.2.

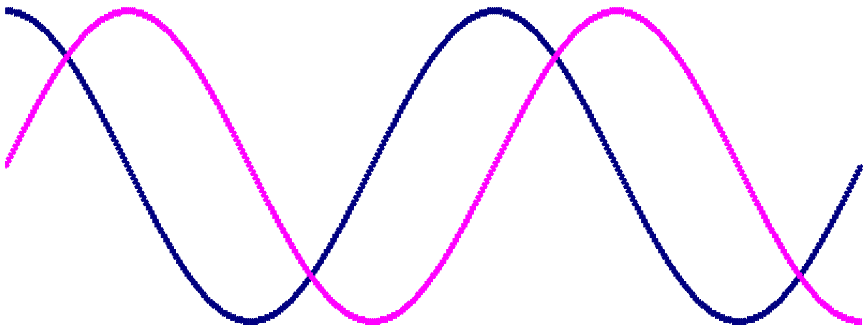


Figure A4.2: Sinusoidal Encoder Signals

When the DMC-40x0 is ordered with the ICM-42100, the position is tracked on two levels. First, the number of coarse cycles is counted much like is done with a quadrature encoder. On the fine level the precise position inside

the cycle is determined from the two sinusoidal signals using bit-wise interpolation. This interpolation can be set by the user in the range of 2^5 through 2^{12} points per sinusoidal cycle via AF command. See the AF command in the command reference for more informaiton.

The unique position within one cycle can be read using the following equation:

$$\text{Fine} = \frac{2^n}{360} \tan^{-1}\left(\frac{V_b}{V_a}\right)$$

The overall position can be determined using:

$$\text{Position} = \text{Coarse\_cycles} \cdot 2^n + \text{Fine}$$

Where:

n is the number of bits of resolution that were used in the conversion.

Coarse_cycles is the whole number of cycles counted.

Fine is the interpolated position within one cycle.

Vb and Va are the two signals as indicated in Figure A4.2.

For example, if the encoder cycle is 40 microns, AF10 results in 2^{10}=1024 counts per cycle, or a resolution of 39 nanometers per count.